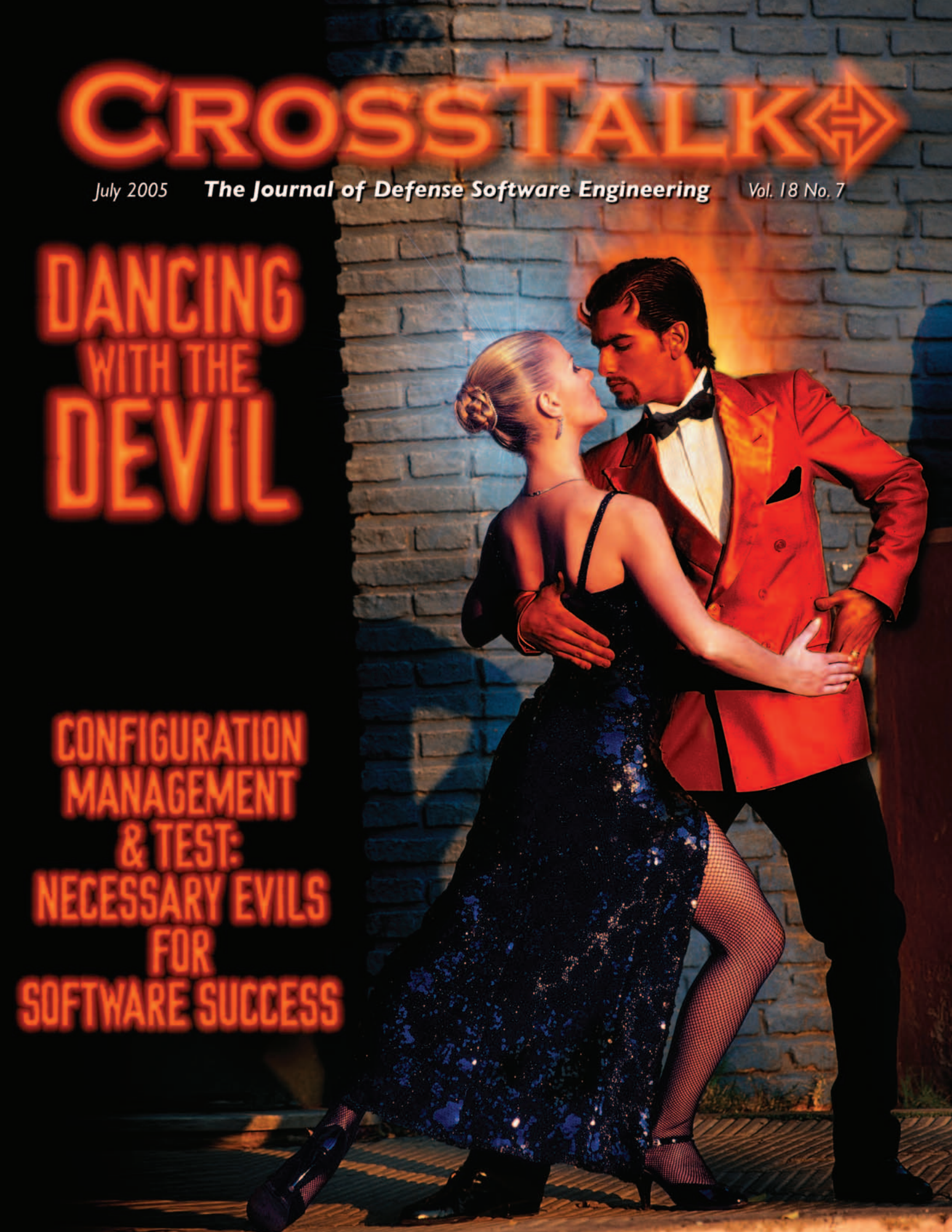
CROSSTALK

July 2005 The Journal of Defense Software Engineering Vol. 18 No. 7

DANCING WITH THE DEVIL

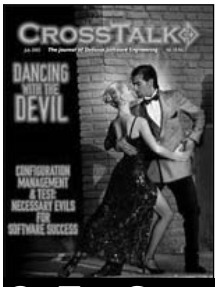
CONFIGURATION MANAGEMENT & TEST: NECESSARY EVILS FOR SOFTWARE SUCCESS



Configuration Management and Test

Software Engineering Technology

Open Forum



ON THE COVER
Cover Design by Kent Bingham.

Departments

CROSSTALK

76 SMXG Co-Sponsor	*Kevin Stamey*
309 SMXG Co-Sponsor	*Randy Hill*
402 SMXG Co-Sponsor	*Tom Christian*
Publisher	*Tracy Stauder*
Associate Publisher	*Elizabeth Starrett*
Managing Editor	*Pamela Palmer*
Associate Editor	*Chelene Fortier-Lozancich*
Article Coordinator	*Nicole Kentta*
Creative Services Coordinator	*Janna Kay Jensen*
Phone	(801) 775-5555
Fax	(801) 777-8069
E-mail	crosstalk.staff@hill.af.mil
CrossTalk Online	www.stsc.hill.af.mil/crosstalk

Oklahoma City-Air Logistics Center (76 SMXG), Ogden-Air Logistics Center (309 SMXG), and Warner Robins-Air Logistics Center (402 SMXG) Software Maintenance Groups (SMXG) are the official co-sponsors of CROSSTALK, The Journal of Defense Software Engineering. The SMXGs and the Software Technology Support Center (STSC) are working jointly to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

The STSC is the publisher of CROSSTALK, providing both editorial oversight and technical review of the journal.

S T S C

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 25.

309 SMXG/MXDB
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820
(801) 775-5555

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CROSSTALK does not pay for submissions. Articles published in CROSSTALK remain the property of the authors and may be submitted to other publications.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CROSSTALK.

Trademarks and Endorsements: This Department of Defense (DoD) journal is an authorized publication for members of the DoD . Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, or the STSC. All product names referenced in this issue are trademarks of their companies.

Coming Events: Please submit conferences, seminars, symposiums, etc. that are of interest to our readers at least 90 days before registration. Mail or e-mail announcements to us.

CrossTalk Online Services: See <www.stsc.hill.af.mil/crosstalk>, call (801) 777-7026, or e-mail <stsc.webmaster@hill.af.mil>.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

From the Sponsor

Processes Provide the Light of Success

The fundamental characteristic of a highly mature software organization is strict adherence to documented processes. Certainly, no processes are more critical to any engineering endeavor's success than well-defined and properly executed configuration management and test processes. Few technical problems lead to more confusion, wasted effort, rework, and overruns than poor configuration management practices. Item identification, version control, traceability, change management, and configuration status accounting should be at least as important to project leadership as tools, techniques, or technical solutions.

Equally important is testing. No matter how meticulously a configuration management effort is developed and prescreening practices are followed, errors will still exist. A sound testing program will reveal these errors. Testers often rely on configuration managers to assist in identifying when and where the errors were inserted. I remember my first introduction to proper configuration management many years ago and how the proverbial lightbulb came on for me. I was amazed that such a basic and critical concept had escaped me to date. Years later, I am still amazed to see this essential project management principle all too frequently ignored and overlooked.

This issue of CROSSTALK addresses the value of configuration management and test and various implementation methods that I hope you will find helpful. Perhaps if your configuration management and test light bulb has not yet been lit, you might at least receive a little glow from your reading time.

Randy B. Hill

Ogden Air Logistics Center, Co-Sponsor

From the Publisher

New Ways to Implement CM and Testing

This month we focus on two very important software engineering disciplines: configuration management (CM) and test. We are continually asked to feature articles on these two disciplines that play an ever-increasing critical role in software development and sustainment.

We begin with *Implementing Configuration Management for Software Testing Projects* by Steve Boycan and Dr. Yuri Chernak, who give a good example of how CM can aid testing projects with the necessary control of evolving testing artifacts. Next, we feature *Configuration Management Fundamentals* by the Software Technology Support Center, a good reminder of the basics of effective CM. In *Finding CM in CMMI,* Anne Mette Jonassen Hass shares her ideas for companies facing the task of improving CM. In *A Correlated Strategic Guide for Software Testing*, Christopher L. Harlow and Dr. Santa Falcone present an iterative testing strategy that has been used in an actual large-scale military software acquisition. Its early testing results can be a guide to later iterative testing in product development cycles. In "*But the Auditor Said We Need to …" Striking a Balance Between Controls and Productivity* by Greg Deller, learn how project teams can better understand auditors' perspectives and recommendations. Finally, don't miss the highlights on pages 16 and 17 from another successful Systems and Software Technology Conference.

I give a special thanks to the authors contributing to this month's issue with their helpful reminders and new insight into practicing effective CM and test.

Tracy L. Stauder

Publisher

Note of Appreciation: We at CROSSTALK extend our appreciation to Walt Lipke as he retired from government service at the end of June. Throughout the years, Mr. Lipke has been a great supporter of CROSSTALK via the articles he shared and his key role in helping us secure funds until our new sponsors started their support. We will miss Mr. Lipke's support in the government sector, but look forward to more words of wisdom via future CROSSTALK articles.

Implementing Configuration Management for Software Testing Projects©

Steve Boycan
Securities Industry Automation Corporation

Dr. Yuri Chernak
Valley Forge Consulting, Inc.

This case study presents the Application Scripting Group's experience in implementing the configuration management (CM) process for critical software testing projects. The article describes the company's test process management objectives and how implementing the CM process helped testers better achieve them. The authors define the types and purposes of the test process milestones and the corresponding types of test model baselines, and describe the CM process implementation with the Rational ClearCase tool.

This case study discusses software testing of use-case-driven projects by the Application Scripting Group (ASG) of the Securities Industry Automation Corporation (SIAC). This SIAC group is responsible for testing highly critical systems used for equity trading at the New York Stock Exchange (NYSE). The high criticality of SIAC systems requires that the software testing process be well planned and well managed. In addition, an important management objective is the continuous improvement of test process performance that focuses on test effort estimation, completeness of test designs before test execution, traceability of test designs to use cases, testing effectiveness in finding defects, and test artifact maintainability and reusability from one project cycle to another. This means that when a given project has ended, a project team has to analyze the actual process performance, perform causal analysis of process deficiencies, and identify improvements for the next project cycle.

To analyze test process performance, testers typically review and analyze the test process artifacts produced and used during a project cycle. However, these testing artifacts, along with their related use cases, evolve during a project cycle and can frequently have multiple versions by project end. Hence, analysis of the process performance from different perspectives requires that testers know exactly which versions of artifacts they used for different tasks. For example, to analyze why the test effort estimates were not sufficiently accurate, testers need the initial versions of use cases, test analysis, and test design specifications they used as a basis for the effort estimation. In contrast, a causal analysis of software defects missed in testing requires testers to have the latest versions of use cases, test analysis, and test design specifications used in test execution.

As this case study illustrates, implementing a configuration management (CM) process provides an effective solution to this issue. It allows testers to capture the versions of their artifacts and the related versions of use cases to produce configuration baselines that can effectively support the analysis of test process performance after the project cycle has ended. In addition, during a project cycle this kind of CM process provides management with much better visibility into and control over the test process. This article describes how the ASG defined their CM process and implemented it for use-case-driven projects with IBM's Rational ClearCase tool. However, the discussed CM process is generic and can be implemented with any CM tool available on the market.

Defining the Test Process Workflows

ASG's implemented CM process was intended to support the test process and make it more efficient and better controlled. Hence, before discussing the CM process, we need to explain how we defined the test process. The Rational Unified Process (RUP) methodology [1] defines the test process as one of its nine disciplines. When testers deal with complex use-case models, they can benefit from further decomposing the RUP's test discipline into six separate workflows shown in Figure 1 and defined in the following paragraphs. As we found on our projects, such test process decomposition can help software testers better cope with functional complexity of software systems, and it can help management better control a testing project.

Workflow 1: Test Analysis and Planning

- **Purpose:** The objectives of this workflow are to define the test strategy and testing objectives for each level of testing, to analyze the use-case model to determine the testing scope and priorities, to provide test effort estimates, to allocate project resources, to analyze quality risks within the context of use-case scenarios, and to develop test ideas about what must be tested for each use case.
- **Key Resulting Artifacts:** A test project plan, system test plan, test automation plan, test guidelines, and test analysis specifications.

Workflow 2: Testware Design and Maintenance

- **Purpose:** The objectives of this workflow are to refine test ideas about what must be tested for each use case and provide details about how to execute these tests. Also, this workflow includes maintenance of the existing test designs.
- **Key Resulting Artifacts:** Test analysis, test design, and test case specifications (and/or test procedure specifications).

Workflow 3: Test Preparation

- **Purpose:** The objectives of this workflow are to set up a test environment and a defect tracking system, generate required test data, and develop test supporting utilities (if required).
- **Key Resulting Artifacts:** Requirements for the test environment, guidelines for test data generation and management, the actual test environment and test data ready for use, and test supporting utilities (if required).

Workflow 4: Test Execution and Reporting

- **Purpose:** The objectives of this

workflow are to execute tests and evaluate the quality of the software product, find and report software defects, and report the product's testing progress and status.
- **Key Resulting Artifacts:** Test analysis and test design specifications (enhanced, for example, with *exploratory* test ideas), software defect reports, test execution logs, and test execution status reports.

Workflow 5: Test Process Evaluation
- **Purpose:** The objectives of this workflow are to evaluate the test process completeness, effectiveness, and efficiency; perform a causal analysis of *test escapes*[1]; and provide recommendations for the test process improvement.
- **Key Resulting Artifacts:** A test summary report, collected test process and product metrics, and a process improvement report (that can include findings of the test escape causal analysis and post-implementation review).

Workflow 6: Regression Test Automation
- **Purpose:** The objectives of this workflow are to develop the test automation architecture and automated regression scripts.
- **Key Resulting Artifacts:** Test automation documentation and test automation software, i.e., automated regression scripts.

Workflows in the Project Cycle
These six test process workflows can, as do all other RUP workflows, overlap in time and iteratively evolve throughout the project cycle (see Figure 1). For example, ASG testers are expected to continue exploring the software product during test execution to develop additional test ideas. Hence, the Test Analysis and Planning and Testware Design and Maintenance workflows largely overlap with the Test Execution and Reporting workflow for this reason.

We have already mentioned three types of test documentation used on ASG projects: test analysis, test design, and test case specifications. Now, we need to explain their purposes. The test analysis specification is developed for each use case. It provides analysis and decomposition (slicing) of a given use case and identifies its quality concerns to be addressed in testing. The test design document is also developed for each use case. It captures a high-level testing logic and rationale for test case selection for each of the quality concerns identified in the corresponding test analysis specification. On ASG projects, testers primarily use the test design specifications for manual test execution. Lastly, the test case specifications focus on the how to execute testing details and are primarily intended for the test automation personnel that use them as functional specifications for developing automated regression scripts.

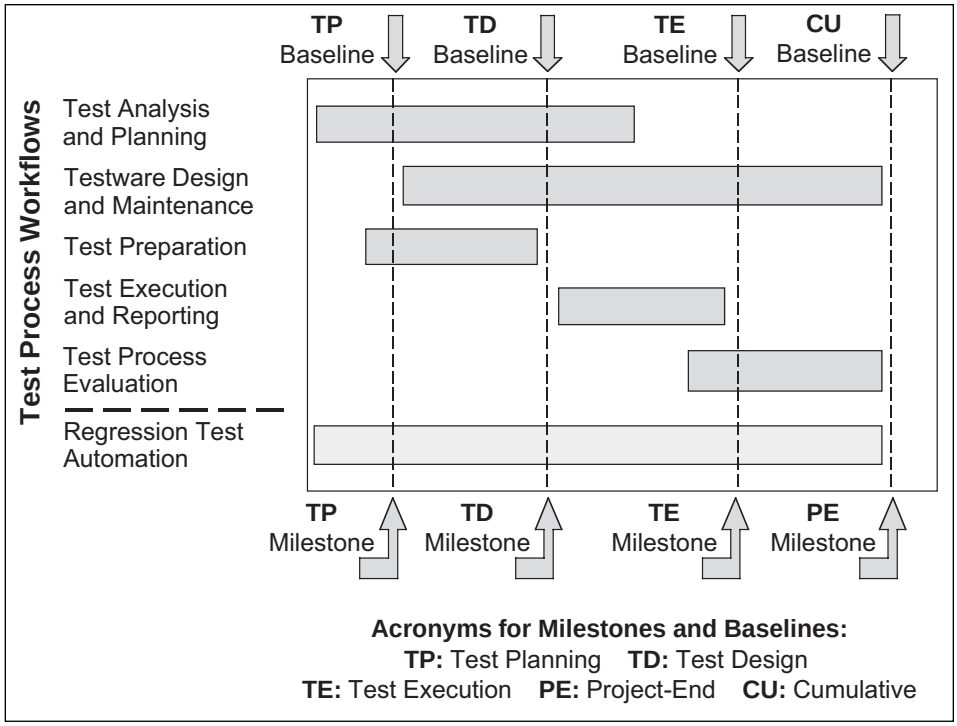


Figure 1: *Project Milestones and Related Baselines*

As Figure 1 shows, the Regression Test Automation workflow spans the entire testing project. In our case, it is managed as a separate project with its own project plan and personnel possessing specialized programming skills. The artifacts of this workflow, i.e., test automation documentation and regression scripts, are also considered a part of the entire test model, which is a collection of all testing artifacts [1].

Defining the Configuration Management Process
According to our CM process, test model baselines are intended to support the test process milestones. For this reason, there is a one-to-one relationship between the baselines and milestones as Figure 1 shows. This section discusses how we defined the types of test project milestones and their corresponding baselines.

Test Project Milestones
The term *milestone* sometimes has different meanings on different projects. According to Webster's dictionary, milestone is defined as "a significant point in development." In the case of ASG, a project milestone means a significant point in the test project life cycle where a test team completes a critical task and makes an important project decision. Thus, ASG defined its project milestones as follows:
- **Test Planning (TP) Milestone.** The test team decides that the test project is feasible and confirms the test project plan and schedule. Testers make this decision based on the completed test analysis and test effort estimation.
- **Test Design (TD) Milestone.** Testers decide that they have completed their test designs sufficiently enough to start test execution.
- **Test Execution (TE) Milestone.** The test team decides to stop testing. Testers make this decision based on evidence that they have met the defined test exit criteria.
- **Project-End (PE) Milestone.** The test team finishes the project cycle and decides which of the test model artifacts should be maintained and reused in future project cycles.

As we already mentioned, the decisions made at project milestones will be evaluated later as part of the project performance improvement task. Hence, testers need to capture the versions of the test model artifacts that supported each of the project milestones during a project cycle. An identified and fixed configuration of these versions is called

Test Model Artifacts	TP Baseline	TD Baseline	TE Baseline	CU Baseline
Component 1- Manual Testing Artifacts				
Test Plan	X	X	X	
Test Analysis Specifications	X	X	X	X
Test Design Specifications		X	X	X
Test Execution Logs			X	
Test Summary Report			X	
Baseline Status Reports	X	X	X	X
Component 2 - Automated Testing Artifacts				
Automation Project Plan	X			X
Test Case Specifications				X
Script Design Specifications				X
Automation Infrastructure Designs				X
Automated Regression Scripts				X
Automation Infrastructure Application Program Interfaces				X

Table 1: *Mapping the Test Model Artifacts to Their Baseline Types*

baseline, and the next section discusses how ASG defined the baseline types.

Test Model Baselines

ASG defined the following types of test model baselines:

- **TP Baseline.** This baseline supports the TP milestone. It captures the version of a test plan that identifies use cases in the scope of the testing project. Also, it captures the versions of test analysis specifications that testers use as a basis for the test effort and schedule estimation.
- **TD Baseline.** This baseline supports the TD milestone. It provides evidence that testers sufficiently completed test designs and can start test execution. It captures the versions of test analysis and test design specifications that have been completed based on use cases and other available functional specifications, and that have been updated based on the peer-review findings. In addition, this baseline includes the latest version of a test plan document.
- **TE Baseline.** This baseline supports the TE milestone. It captures information about how the system was tested that testers deem necessary to support their conclusion about the software product's quality. In particular, it includes the latest versions of test analysis and test design specifications[2] that have evolved during the manual test execution. In addition, it includes the test execution logs, both manual and automated, and a test summary report.
- **Cumulative (CU) Baseline.** This baseline supports the PE milestone. It captures test assets intended for maintenance and reuse in subsequent project cycles. This baseline has two components: manual testing artifacts and automated testing artifacts. Unlike the other baseline types that capture artifacts created and used only in a given project cycle, the CU baseline captures the cumulative set of artifacts created both in the current project cycle and in previous cycles[3].

Finally, different baseline types can be composed of different test model artifacts. Table 1 shows the mapping between the test artifacts and their corresponding baseline types established for the ASG projects.

Implementing the CM Process

This section discusses the CM process reference model, which was selected for the ASG projects, and explains in detail the CM process implementation.

A CM Process Reference Model

ASG implemented the CM process by following the practices of the Software Configuration Management Key Process Area defined in the Software Engineering Institute's Capability Maturity Model® (CMM®) framework [2]. To better manage the CM process implementation, we further grouped the CMM practices by four categories that compose the conventional CM discipline [3]:

- Configuration Identification.
- Configuration Control.
- Configuration Status Accounting.
- Configuration Auditing.

The implementation of the CM process began by producing a general document for the department that defined a CM policy, glossary of CM terms, a standard CM process, and guidelines for its implementation. Then, based on this document, each project team developed its own CM plan document that followed the Institute of Electrical and Electronics Engineers (IEEE) Std.828-1998 "IEEE Standard for Software Configuration Management Plans." These CM plans were project-specific and defined configuration items and their naming conventions, the CM repository structure, and the team member roles and responsibilities. In particular, each project assigned the CM manager role to one of the team members. Finally, in implementing the CM process, the project teams used the CM plans as a basis for performing their CM activities.

Configuration Identification

When performing the configuration identification activities, a project team decides which test model artifacts must be under configuration control, what should be in a team-shared repository, and how the repository should be structured. On use-case-driven projects, use-case models can form complex structures via the include/extend relationships [4]. In this case, different testers can be assigned to produce test designs for related use cases. Hence, it is important that team members have quick access in the course of a project to the latest versions of each other's test documentation. Establishing a common CM repository of the test model artifacts provides an effective solution to this issue.

ASG projects created their CM repositories using IBM's Rational ClearCase tool. Each repository contained a set of directories, each storing a particular type of configuration item. For example, all test design specifications, created by the project team, were stored in the same directory. Table 1 shows the artifacts of the test model that we included in the configuration control. As you can see, they are logically divided into two components: manual testing artifacts and automated testing artifacts that, in turn, include the automation design documentation and software, i.e., automated regression scripts.

ASG then defined a naming convention for all artifacts under configuration control. Here is a file name example of a test design specification that illustrates our naming convention: TDS_DB_9.1 .doc. In this example, the file name is composed of the following parts:

- **Artifact type:** TDS, meaning the arti-

® Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

fact type *test design specification*.
- **Project code:** DB, meaning the project name *Display Book*.
- **Use Case ID number:** 9.1.

By our convention, use cases and test design specifications have a one-to-one relationship. Hence, by including the use case number in the test design file name, ASG established an explicit traceability from use cases to their corresponding test designs.

Configuration Control

Performing the configuration control activities involves controlling change requests for configuration items, reporting and tracking problems, controlling versions of configuration items, capturing the change history each time a new version is created, labeling artifacts of a test model, and creating its baselines associated with various project milestones. A project manager is the primary source of change requests. He/she informs the testers when a new project cycle begins, when they should start working on test designs, which new automated scripts should be created, etc. In addition, any project team member who finds a problem with automated scripts reports it via a defect tracking system.

The version control of test project artifacts was handled by the CM tool that allows the creation of new versions of a given artifact and the capture of its change history notes, the date/time when the new version was created, the owner of this version, and so on. For some of the artifact types, ASG also created custom attributes in ClearCase[4]. For example, for the test design specifications we created attributes to capture the peer-review date and the total number of test cases for a given test design. This information provided management with better visibility into the state of evolving test designs. Also, the number of test cases was captured as part of the historical project data for evaluating and improving the test execution effort estimation.

Creating a test model baseline is a three-step process. First, the CM manager creates labels that correspond to the project milestones discussed earlier. Second, the artifact versions intended for inclusion in a particular baseline are labeled to correspond to the requested baseline. For the TP and TD baselines, the artifact owner is responsible for labeling versions of his/her own work products, as only the owner knows when an artifact is ready for inclusion in either the TP or TD baseline. In contrast, applying labels at the end of test execution (TE baseline) or at the end of the entire project cycle (CU baseline) can be done for all required artifacts at the same time. Hence, in the case of TE and CU baselines, the CM manager can be responsible for labeling the test model artifacts by simply running a script. Finally, at the third step the CM manager locks the label and completes the baseline creation by producing and publishing a baseline status report. Following these steps allows creating the required baselines and capturing the versions of evolving test model artifacts at different points in the project life cycle.

Configuration Status Accounting

The main objective of the ASG status accounting activities was to verify and report to a team and its management that a given baseline is compliant with its completion criteria. The CM manager was responsible for this task. First, before creating each baseline, this task required verifying that the set of test model artifacts was compliant with the baseline completion criteria. Second, after a given baseline has been created, the task required producing and publishing a baseline status report. As different test model baselines have different purposes and different contents shown in Table 1, ASG defined the completion criteria for each of them as follows:

"By following the defined CM process, our testing teams established their team-shared project repositories, implemented effective version control of their artifacts, and produced test model baselines to support different project milestones."

TP Baseline

The TP baseline (*captures information used for the test effort estimation*) requires the following:
- A test plan has been reviewed and its current version has been labeled.
- All required test analysis specifications exist in the CM repository.
- The latest versions of test analysis specifications, used for the test effort estimation, have been labeled and refer to the corresponding use-case document versions.
- A baseline status report has been created and labeled.

TD Baseline

The TD baseline (*captures test designs to be used for test execution*) requires the following:
- The latest version of the test plan document has been labeled.
- All required test analysis and test design specifications have been completed and peer-reviewed.
- The latest versions of test analysis and test design specifications, ready for test execution, have been labeled.
- All labeled versions of test analysis and test design specifications refer to the corresponding use-case versions.
- All labeled test design specifications capture the peer-review date and the total number of test cases.
- A baseline status report has been created and labeled.

TE Baseline

The TE baseline (*captures test artifacts actually used for test execution*) requires the following:
- The latest version of the test plan document has been labeled.
- The latest versions of test analysis and test design specifications, actually used for test execution, have been labeled.
- All labeled versions of test analysis and test design specifications refer to the corresponding use-case versions.
- All labeled test design specifications capture the total number of test cases (that may have increased during test execution).
- The test execution logs (manual and automated) exist in the CM repository and their latest versions have been labeled.
- The test summary report exists in the CM repository and its latest version has been labeled.
- A baseline status report has been created and labeled.

CU Baseline

The CU baseline (*captures test artifacts intended for reuse and maintenance*) requires the following:
- The latest versions of all selected test

analysis specifications have been labeled.
- The latest versions of all selected test design specifications have been labeled.
- The latest version of the test automation plan has been labeled.
- The latest versions of all selected test case specifications have been labeled.
- The latest versions of test automation documentation have been labeled.
- The latest versions of automated scripts that have been accepted for production have been labeled.
- Each accepted automated script captures the date it was accepted for production and the name of a team member who tested and accepted the script.
- A baseline status report has been created and labeled.

Configuration Auditing

Baseline auditing was an important part of our CM process. Especially in the beginning of the process implementation a risk existed that some team members, because of their lack of experience, might not follow the process exactly as defined. To ensure that the CM process is properly followed, the ASG department established a quality assurance (QA) function. The QA personnel oversaw all of the process improvement tasks in the department, including the CM process implementation and project baseline audits.

Each testing project plan included baseline audit tasks and assigned a QA auditor as a project resource responsible for the task. By having access to each project team's configuration repository, the auditor could verify that each created baseline was compliant with its defined completion criteria. In addition, the auditor was monitoring each project team's CM activities during a project cycle. Thus, any deviations from the process then could be identified and corrected earlier in the project. The audit findings were reported to project teams and discussed at their status meetings. Also, the audit summary report was periodically submitted to and reviewed by the department head.

CM Process Implementation Challenges

As is the case with any software process improvement implementation, while implementing our new CM process we experienced a few challenges. These challenges can be described from two perspectives: (1) a general process improvement perspective, and (2) a CM process-specific perspective.

From the general process improvement perspective, the success of any new process implementation depends not only on a good definition of the process activities and procedures, but also on a number of other (process supporting) critical factors. Among them, the most important are establishing process ownership and leadership, allocating necessary resources, personnel training, management reviews, and process auditing. Because of this dependency, we found that the practices defined in the CMM as the following common features [2] – c*ommitment to perform, ability to perform, measurement and analysis, and verifying implementation* – are all equally important for successful process implementation as the practices in the main CMM category – *activities to perform*.

From the CM process-specific perspective, before a project team starts implementing a new CM process, it should decide how to handle legacy test model artifacts when moving them into the new CM repository and creating an initial project baseline. Likely, these artifacts will not be in compliance with the defined CM process requirements. For example, old test designs might not have their peer review dates and/or references to their corresponding use-case versions. Likewise, old automated scripts might not have their acceptance dates, etc. Hence, a newly created baseline might not satisfy its completion criteria when it includes the legacy configuration items.

One way to resolve this issue is to establish a convention defining how the missing metadata can be substituted while moving the legacy artifacts to the new CM repository. This way a status report – used to determine the baseline's completion – will not have blank values for the metadata pertinent to the legacy configuration items.

Conclusion

This case study discussed our experience with implementing the CM process for highly critical software testing projects. An important management objective on our projects is to establish a framework for (a) effective control of testing projects, and (b) continuous analysis and improvement of test process performance. As we illustrated in this article, implementing the CM process allows software testers to better achieve this management objective. By following the defined CM process, our testing teams established their team-shared project repositories, implemented effective version control of their artifacts, and produced test model baselines to support different project milestones. These baselines captured configurations of versions of the testing artifacts and their related use cases that could later support the testers' analysis and improvement of test process performance. Finally, the discussed CM process is generic and can be implemented with any available CM tool.◆

References

1. Kroll, P., and P. Kruchten. The Rational Unified Process Made Easy: A Practitioner's Guide to the Rational Unified Process. Addison-Wesley Professional, 2003.
2. Paulk, M., et al. The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley Professional, 1995.
3. Ben-Menachen, M. Software Configuration Management Guidebook. McGraw-Hill Book Company, 1994.
4. Bittner, K., and I. Spence. Use Case Modeling. Addison-Wesley, 2003.

Notes

1. From a system test perspective, *test escape* is a software defect missed in system testing and found in a later stage, for example, during independent quality assurance testing or in production.
2. Capturing the versions of test designs at this point is critical to supporting at a later time the causal analysis of test escapes, especially those found in production.
3. Regression Test Automation workflow can have its own intermediate baselines; however, the versions of these workflow deliverables must be synchronized with other test model artifacts at the end of a project cycle (in the CU baseline)
4. This ClearCase feature – adding custom attributes to configuration items – may not be available in some commercial CM tools.

Acknowledgements

The authors are grateful to the CROSSTALK reviewers and Robin Goldsmith at GoPro Management for their feedback and comments that helped us improve this article. Our special thanks to the ASG testers who have implemented and followed the CM process discussed in this article, and helped us refine the process and make it effective.

About the Authors

Steve Boycan is managing director for Process Improvement and Software Engineering and Testing Support at Securities Industry Automation Corporation where he manages the process improvement program and various testing activities for critical New York Stock Exchange trading systems. He has been leading software process improvement efforts in the military, telecommunications, and financial sectors for the last 10 years. As a certified Capability Maturity Model® (CMM®) Lead Assessor, he has performed a number of CMM-based assessments and provided process improvement guidance for various information technology organizations.

Securities Industry Automation Corporation
2 MetroTech CTR
Brooklyn, NY 11201
Phone: (212) 383-2963
Fax: (718) 923-6068
E-mail: sboycan@siac.com

Yuri Chernak, Ph.D., is the president and principal consultant of Valley Forge Consulting, Inc. As a consultant, Chernak has worked for a number of major financial firms in New York helping senior management improve their software testing process. Currently, his research interests focus on use-case-driven testing and test process assessment and improvement. Chernak is a member of the Institute of Electrical and Electronics Engineers (IEEE) Computer Society. He has been a speaker at several international conferences and has published papers on software testing in the IEEE publications and other professional journals. Chernak has a doctorate in computer science.

Valley Forge Consulting, Inc.
233 Cambridge Oaks ST
Park Ridge, NJ 07656
Phone: (201) 307-4802
Fax: (201) 307-4803
E-mail: ychernak@yahoo.com

CALL FOR ARTICLES

If your experience or research has produced information that could be useful to others, CROSSTALK can get the word out. We are specifically looking for articles on software-related topics to supplement upcoming theme issues. Below is the submittal schedule for three areas of emphasis we are looking for:

Total Creation of a Software Project
December 2005
Submission Deadline: July 18

Communications
January 2006
Submission Deadline: August 22

What Is Up and Coming?
February 2006
Submission Deadline: September 19

Please follow the Author Guidelines for CROSSTALK, available on the Internet at <www.stsc.hill.af.mil/crosstalk>. We accept article submissions on all software-related topics at any time, along with Letters to the Editor and BackTalk.

COMING EVENTS

August 15-17
International Conference on Information Reuse and Integration: Knowledge Acquisition and Management
Las Vegas, NV
www.cs.fiu.edu/IRI05

August 16-18
ICSEng '05 International Conference on Systems Engineering
Las Vegas, NV
www.icseng.info

August 22-26
Association for Computing Machinery SIGCOMM 2005
Philadelphia, PA
www.acm.org/sigs/sigcomm/sigcomm2005/index.html

September 12-16
Practical Software Quality and Testing Conference 2005 North
Minneapolis, MN
www.psqtconference.com/2005north

September 18-23
International Function Point Users Group 1st Annual International Software Measurement and Analysis Conference
New Orleans, LA
www.ifpug.org/conferences/annual.htm

September 19-22
Better Software Conference and Expo 2005
San Francisco, CA
www.sqe.com/bettersoftwareconf

September 26-27
PDF Conference and Expo
Washington, DC
www.pdfconference.com

May 1-4, 2006
2006 Systems and Software Technology Conference
STC Systems & Software Technology Conference
Salt Lake City, UT
www.stc-online.org

Configuration Management Fundamentals

Software Technology Support Center

The U.S. Air Force's Software Technology Support Center offers an updated and condensed version of the "Guidelines for Successful Acquisition and Management of Software-Intensive Systems" (GSAM) on its Web site <www.stsc.hill.af.mil/resources/tech_docs>. This article is taken from Chapter 9 "Configuration Management" of the GSAM (Ver. 4.0). We are pleased that all editions have been so well received and that many individuals and programs have worked hard to implement the principles contained therein. The latest edition provides a usable desk reference that gives a brief but effective overview of important software acquisition and development topics, provides checklists for rapid self-inspection, and provides pointers to additional information on the topics covered.

Change is a constant feature of software development. To eliminate change is to remove the opportunities to take advantage of lessons learned, to incorporate advancing technology, and to better accommodate a changing environment. Refusal to incorporate change can mean system limitations and early obsolescence, which, in the world of technology, can sign your system's death certificate before it is born. However, change is not universally benign and must be controlled in its introduction to a project.

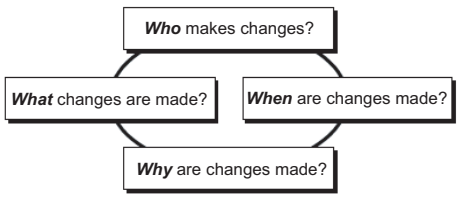
All projects change something. As a project is executed, changes to the initial project plan and products are a natural occurrence. The following are common sources of changes:

- Requirements. The longer the delivery cycle, the more likely they will change.
- Changes in funding.
- Technology advancements.
- Solutions to problems.
- Scheduling constraints.
- Customer expectations.
- Serendipitous (unexpected) opportunities for an improved system.

Some of these changes may appear as options while others may be mandated from above or by circumstance, as in the loss of funding. While all progress is accompanied by change, not all change is indicative of progress. If not properly handled, change can slip the schedule, affect the quality, and even kill the project. As a project draws closer to its completion, the impacts of change are more severe [1]. Clearly, a mechanism is needed to control change.

In software development and other projects, proposed changes must be evaluated to determine their overall contribution to the project goals. Do they lead to improvements or do they ultimately impede or lower project quality? Even those changes that are ultimately beneficial must be controlled in their introduction and implementation.

Figure 1: *Configuration Management Questions*



Putting a bigger engine in a plane may improve its capabilities, but it cannot be implemented until the aircraft's structure has been found capable or been upgraded to support the increased weight and thrust.

Configuration management (CM) is the process of controlling and documenting change to a developing system. It is part of the overall change management approach. As the size of an effort increases, so does the necessity of implementing effective CM. It allows large teams to work together in a stable environment while still providing the flexibility required for creative work [2]. CM in a software environment is an absolute necessity. CM has three major purposes [1]:

1. Identify the configuration of the product at various points in time.
2. Systematically control changes to the configuration.
3. Maintain the integrity and traceability of the configuration throughout the product life cycle.

CM accomplishes these purposes by answering and recording the answers to the change questions: who, what, when, and why, shown in Figure 1 [1]. Being able to answer these questions is a sign of effective CM.

Effective CM provides the following essential benefits to a project:

1. Reduces confusion and establishes order.
2. Organizes the activities necessary to maintain product integrity.
3. Ensures correct product configurations.
4. Limits legal liability by providing a record of actions.
5. Reduces life-cycle costs.
6. Enables consistent conformance with requirements.
7. Provides a stable working environment.
8. Enhances compliance with standards.
9. Enhances status accounting.

In short, CM can provide cost effective project insurance when properly planned, organized, and implemented. It must be integral to your overall project execution, and to your charter/customer agreement. Proposed changes must be dealt with systematically, promptly, and honestly [1]. If the CM process is unreasonable or unresponsive, people will try to circumvent the process, leading to chaos and a loss of the benefits of true CM.

Process Description

While CM is a major element of a change control program, it is such a multifaceted discipline that it should be considered not simply as another activity, but as a program in and of itself. Establishing an effective CM program requires an understanding of CM functions and of the overall CM process.

Functions of Configuration Management

CM is comprised of four primary functions: identification, change control, status accounting, and auditing. These are shown in Figure 2, along with their subfunctions. All CM activity falls within the bounds of these functions.

Identification

This function identifies those items whose configuration needs to be controlled, usually consisting of hardware, software, and documentation. These items would probably include such things as specifications, designs, data, documents, drawings, software code and executables, components of the software engineering environment (compilers, linkers, loaders, hardware environment, etc.), and hardware components and assem-

blies. Project plans and guiding documents should also be included, especially the project requirements. A schema of names and numbers is developed for accurately identifying products and their configuration or version level. This must be done in accordance with project identification requirements. Finally, a baseline configuration is established for all configuration items and systems. Any changes to the baseline must be with the concurrence of the configuration control organization [2].

Although key components to be managed are requirements and source code, related documentation and data should be identified and placed under CM control. It is important to store and track all environment information and support tools used throughout the software life cycle to ensure that the software can be reproduced. Table 1 lists examples of items typically put under CM control.

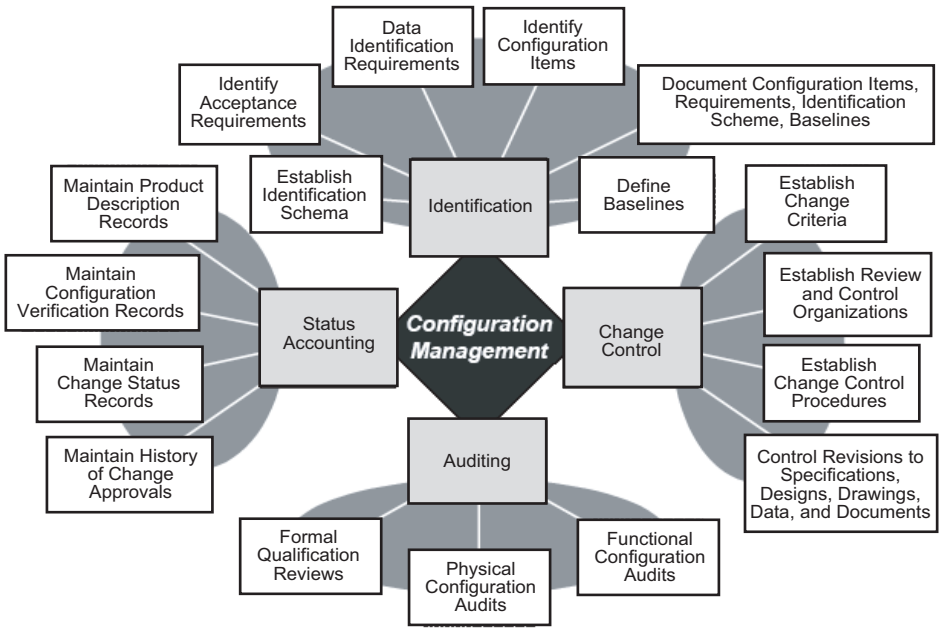


Figure 2: *Major Functions of Configuration Management [2]*

Change Control

Configuration control establishes procedures for proposing or requesting changes, evaluating those changes for desirability, obtaining authorization for changes, publishing and tracking changes, and implementing changes. This function also identifies the people and organizations who have authority to make changes at various levels (configuration item, assembly, system, project, etc.,) and those who make up the configuration control board(s) (CCB). (According to IEEE 610.12 [3], a CCB is a group of people responsible for evaluating and approving or disapproving proposed changes to configuration items, and for ensuring implementation of approved changes.)

Additionally, various change criteria are defined as guidelines for the control organizations. Different types of configuration items or different systems will probably need different control procedures and involve different people. For example, software configuration control has different needs and involves different people than communications configuration control and would probably require different control rules and a different control board [2]. Configuration change control activities include the following:

- Defining the change process.
- Establishing change control policies and procedures.
- Maintaining baselines.
- Processing changes.
- Developing change report forms.
- Controlling release of the product.

A generic software change process is identified in Figure 3 (see next page).

Status Accounting

Status accounting is the documentation function of CM. Its primary purpose is to maintain formal records of established configurations and make regular reports of configuration status. These records should accurately describe the product, and are used to verify the configuration of the system for testing, delivery, and other activities. Status accounting also maintains a history of change requests and authorizations, along with status of all approved changes. This includes the answers to the CM questions in Figure 1 [2].

Key information about the project and configuration items can be communicated to project members through status accounting. Software engineers can see what fixes or files were included in which baseline. Project managers can track completion of problem reports and various other maintenance activities. Minimal reports to be completed include transaction log, change log, and item *delta* report. Other typically common reports include resource usage, *stock status* (status of all configuration items), changes in process, and agreed-upon deviations [6].

Auditing

Effective CM requires regular evaluation of the configuration. This is done through the auditing function, where the physical and functional configurations are compared to the documented configuration. The purpose of auditing is to maintain the integrity of the baseline and release configurations for all controlled products [2]. Auditing is accomplished via both informal monitoring and formal reviews.

Configuration auditing verifies that the software product is built according to the requirements, standards, or contractual agreement. Test reports and documentation are used to verify that the software meets the stated requirements. The goal of a configuration audit is to verify that all software products have been produced, correctly identified and described, and

Table 1: *Items Under CM Control*

Items Under CM Control	
System data files	Source code modules
Requirements specifications	System build files/scripts
Design specifications	Interface specifications
Test plans	Software architecture specifications
Test data sets	Test procedures
User documentation	Test results
Quality plans	Software development plan
Compilers	Configuration management plans
Debuggers	Linkers and loaders
Shell scripts	Operating systems
Other related support tools	Third-party tools
Development procedures and standards [4]	Procedure language descriptions

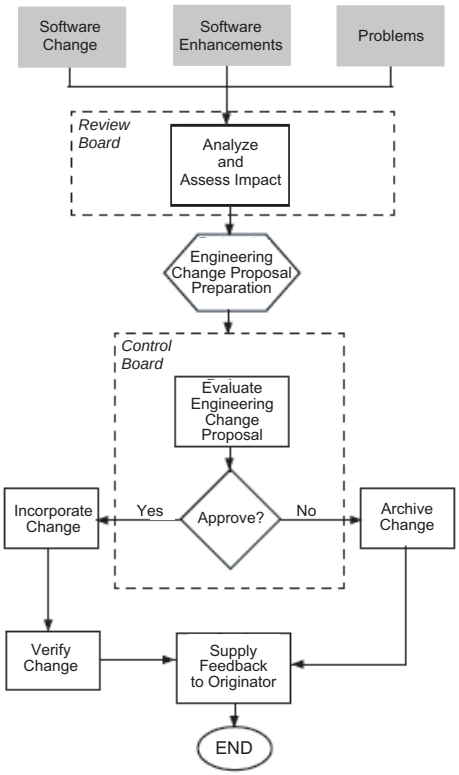


Figure 3: *Generic Change Process* [5]

that all change requests have been resolved according to established CM processes and procedures. Informal audits are conducted at key phases of the software life cycle.

There are two types of formal audits that are conducted before the software is delivered to the customer: Functional Configuration Audit (FCA) and Physical Configuration Audit (PCA). FCA verifies that the software satisfies the software requirements stated in the System Requirements Specification and the Interface Requirements Specification. In other words, the FCA allows you to validate the system against the requirements. The PCA determines whether the design and reference documents represent the software that was built. Configuration audit answers the questions, "Does the system satisfy the requirements?" "Are all changes incorporated in this version?" Configuration audit activities include the following:

- Defining audit schedule and procedures.
- Identifying who will perform the audits.
- Performing audits on established baselines.
- Generating audit reports.

Establishing a Software Baseline Library

In support of the above activities, a software baseline library is established. The library is the heart of the CM system. It serves as the repository for the work products created during the software life cycle. Changes to baselines, and the release of software products, are systematically controlled via the change control and configuration auditing functions. The software library provides the following:

- Supports multiple control levels of Software Configuration Management (SCM).
- Provides for the storage and retrieval of configuration items or units.
- Provides for the sharing and transfer of configuration items or units between control levels within the library.
- Provides for the storage and recovery of archive versions of configuration items or units.
- Helps to ensure correct creation of products from the software baseline library.
- Provides storage, update, and retrieval of CM records.
- Supports production of CM reports.
- Provides for maintenance of library structure [7].

In the past, libraries have been composed of documentation on hard copy and software on machine-readable media. Today, with the advances in information technology and standards that encourage contractors to use automated processing and electronic submittal techniques, organizations are moving toward maintaining all information on machine-readable media.

Configuration Management Process

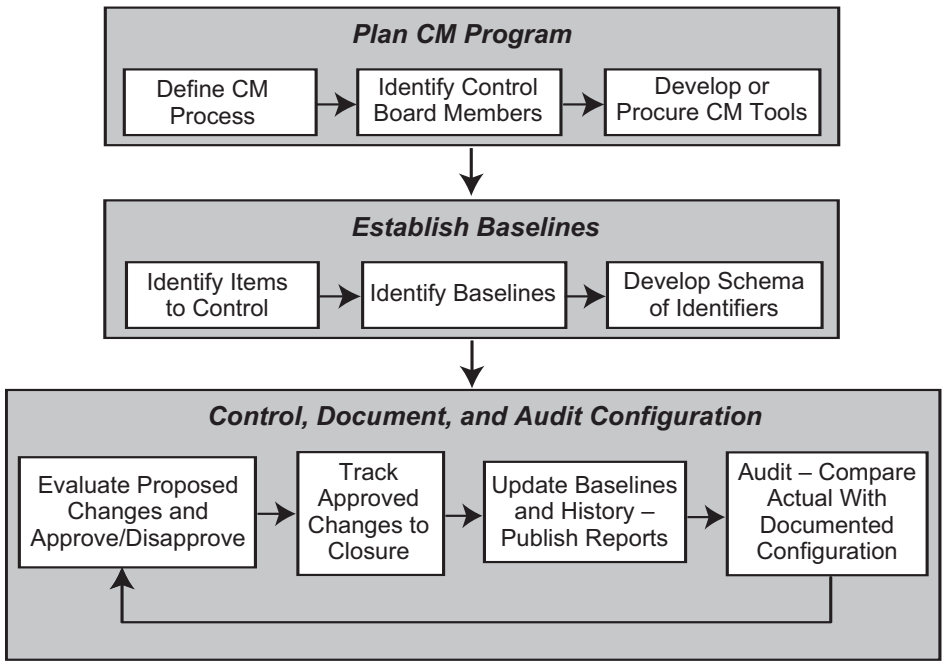
Understanding what CM is supposed to accomplish is one thing. Putting it into practice is another. As with most project activities, CM begins with planning. With a plan, configuration baselines can be established. Following this initial configuration identification, the cyclical configuration control process is put into motion. These three major CM implementation activities are shown in Figure 4.

Planning

Planning begins by defining the CM process and establishing procedures for controlling and documenting change. A crucial action is the designation of members of the CCB. Members should be chosen who are directly or indirectly involved or affected by changes in configuration. For example, a software CCB would obviously be populated with representatives from different software teams, but software affects many more aspects of a project. There should also be representatives from the hardware, test, systems, security, and quality groups as well as representatives from project management and possible other organizations.

Not all changes would be reviewed by this august body. Changes occur at different system levels and affect different portions of the overall system. Many changes will probably only affect a small subset of the system and could therefore be reviewed and approved by a smaller group. Some sort of delineation of change levels should be made during planning to keep change decisions at the proper level. While software CM is essen-

Figure 4: *Configuration Management Implementation Process*



tial, there may need to be other CCBs to control change in other areas of the project. Again, this is something that should be worked out in the planning phase.

Various software tools exist that can facilitate the CM process flow and maintain configuration history. Using a CM software tool is highly recommended. The temptation will be to choose a tool because it looks good in a demonstration and then build the CM process around it. The process should be defined first, and then a tool chosen to facilitate the process.

Establishing Baselines

Once the CM program exists on paper, it must be determined just what configurations it will control. The second major step of implementing effective CM is identifying what items, assemblies, code, data, documents, systems, etc. will fall under configuration control. With the configuration items identified, the baseline configuration must be identified for each item. For items that already exist, it may prove to be nothing more than examining or reviewing and then documenting. For those items that have not been developed yet, their configuration exists in the requirements database or in the project plans. Until they come into physical or software reality, changes to their configuration will consist only of changes to the requirements or plans.

Another essential activity in this step is developing a schema of numbers, letters, words, etc. to accurately describe the configuration revision, or version, for each general type of configuration item. There may be project requirements that dictate some type of nomenclature, or there may be an organizational or industry standard that can be used as the basis for configuration identification.

Controlling, Documenting, and Auditing

When the baselines have been established, the challenge becomes one of keeping the actual and documented configurations identical. Additionally, these baselines must conform to the configuration specified in the project requirements. This is an iterative process consisting of the four steps shown in Figure 4.

All changes to the configuration are reviewed and evaluated by the appropriate configuration control representatives specified in the CM plan. The change is either approved or disapproved. Both approvals and disapprovals are documented in the CM history. Approved changes are published and tracked or monitored until they are implemented. The appropriate configuration baseline is then updated, along with all other applicable documents, and reports are published and sent to affected organizations indicating the changes that have occurred. At selected time intervals and whenever there appears to be a need, products and records are audited to ensure the following:

- The actual configuration matches the documented configuration.
- The configuration is in conformance with project requirements.
- Records of all change activity are complete and up-to-date.

The controlling, documenting, auditing cycle is repeated throughout the project until its completion.

Updating the CM Process

It is unlikely a perfect CM program will be assembled during the initial planning stage. There will be learning and changes in the program that indicate a need for adjustments in the CM process. These may be any mixture of modifications to make it more efficient, responsive, or accurate. When changes in the CM process are needed, consider them as you would any other changes, get the approval of all participating organizations, and implement them as appropriate. It would be ironic indeed to have an unchanging change process.

Configuration Management Checklist

This checklist is provided to assist you in establishing an effective CM program. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

CM Planning

- ❑ Have you planned and documented a configuration management process?
- ❑ Have you identified CCB members for each needed control board?
- ❑ Has CM software been chosen to facilitate your CM process?

Establishing Baselines

- ❑ Have all configuration items been identified?
- ❑ Have baselines been established for all configuration items?
- ❑ Has a descriptive schema been developed to accurately identify configuration items and changes to their configuration?

Controlling, Documenting, and Auditing

- ❑ Is there a formal process for documenting and submitting proposed changes?
- ❑ Is the CCB active and responsible in evaluating and approving changes?
- ❑ Is there a *higher authority* to appeal to when the CCB gets *hung*, and cannot come to a consensus?
- ❑ Are all changes tracked until they are fully implemented?
- ❑ Are all changes fully documented in the baseline documents and change histories?
- ❑ Are regular reports and configuration updates published and distributed to interested organizations?
- ❑ Are regular audits and reviews performed to evaluate configuration integrity?
- ❑ Are configuration errors dealt with in an efficient and timely manner?

Updating the Process

- ❑ Is the CM program itself – its efficiency, responsiveness, and accuracy – evaluated regularly?
- ❑ Is the CM program modified to include recommended improvements when needed?

Case Studies

The following case studies outline specific instances where organizations successfully implemented software CM (SCM).

Selecting a CM Tool

At a large aerospace corporation in the Southeast, the CM manager turned in a recommendation to purchase a CM automated tool that would satisfy all requirements identified by the CM groups. Management delayed acting on the recommendation to give the other engineering departments time to review the recommended tool.

In the end, the recommendation to purchase the tool was cancelled. It was felt that while the tool did support the CM organization, it did not adequately address other developmental considerations that the engineering ranks felt were important. Sometime later a different tool was purchased, one that satisfied all the major requirements of SCM, the software developers, SQA, test, integration, and management organizations.

Overcoming Barriers to SCM

During a recent visit to a private-sector corporation (i.e., they did not deal with government contracts) in New England, it was discovered that the developer's major concern about implementing CM activities was *all the restrictions* they would have to deal with. They had been led to

believe that CM meant formal controls, restricted access, limited ability to apply creative solutions, and so on. When it was suggested that data can transition to formally controlled baselines through a series of informal control steps, and that CM did not mean a lockdown and bottleneck, they became eager to be involved. After a number of meetings, a phased approach to formal CM allowed for the placement of informal controls and data gathering which led to baselined items. Everyone was pleased with the process.

The developers soon realized they could work together with CM as a team to solve problems rather than as two separate organizations with their own concerns and desired solutions. More importantly, perhaps, the CM group learned that when they got out of their corner office and out onto the engineering floor (being support and service oriented) they quickly became an integral part of the engineering and development process and team.

Implementing CM With an Electronic Database

A team of 35 to 40 developers was developing six computer software configuration items, which all had two or more customer variants as well as maintenance variants. The operating system was Unix, and the development languages were Ada, C, and C++. Implementing classical CM in this type of environment would normally require three to four CM practitioners to handle all the code and document manipulations. The team chose instead to implement a mostly developer-executed CM system called *Effective SCM*. They implemented a Software Query Language-compliant, database driven, process oriented CM system, which supports a rule-based, closed-loop, change-package approach to development.

Daily interaction with the CM system by the developers provided 100 percent tracking and status accounting of everything that happened to any file in the systems without the need for intrusion or interference by CM practitioners. The CM practitioners maintained the process model and performed the configured builds. As a result, CM support to this team was less than one person, and in fact was in the order of 80-120 hours per month instead of the more than 400 hours per month that a classical approach would have used.

The electronic database created by the engineers completely documented the execution of their software development plan. It also tracked the history of every file used in the system including change documents, baselines, and releases for each file. Note that rule-based, closed-loop change control electronically implemented business rules that prevented the creation of a new version without authorization and prevented closure of a change request that had not been implemented.

A change-package approach supports electronic creation of new baselines by application of changes to a previous baseline. The tool electronically adds, replaces, or removes files that are related to the list of changes being made and is very effective in tracking development activities. (Note that *Effective SCM* is an unregistered trademark of BOBEV Consulting. For a complete description, see "Effective Software Configuration Management" in CROSSTALK February 1998.)

Lessons Learned

The following are just a sample of the many lessons that have been learned from applying CM and its associated technologies.

The Importance of Planning

With only a few exceptions, if you look at any of the CM standards, manuals, guides, books, etc., you will likely find that CM has four major functions: (1) identification, (2) change control, (3) status accounting, and (4) auditing. In nearly every case, planning is left out. Yet, SCM is using much more complex equipment to establish and maintain complex environments, multiple baselines, multiple environments on multiple platforms, etc. Like everyone else, CM has to do all that faster, cheaper, smarter, and better than before. Planning has become more important than ever.

Planning cannot be interpreted as meaning *a CM Plan has been written*. That is certainly a good start, but much more is needed than just a document that explains SCM's roles and responsibilities. CM planning activities must also include, to name only a few, such things as the following:

- **Metrics.** How long? How many artifacts? When were they created? When were they updated? Where are they?
- **Skill Mix.** What is needed and who has it or who can get it?
- **Infrastructure.** Who is doing what, where, when, and how?
- **Contingencies.** If this happens, then what?
- **Effort Tracking.** Manpower levels.
- **Subcontracts.** Who has responsibility and authority?
- **Resources.** Budget, tool licenses, training, and head count.
- **Matrix Management.** Decentralized work force.
- **Control Transitions.** Informal to formal to field.
- **Records Retention.** What gets kept where and for how long?
- **Control.** Who controls what and how do they do it?
- **Process.** Standardized procedures for repeatability.

Things to Remember

The most significant lessons are the following:

- Get an inside person on your side – an internal champion. They will become an evangelist for your solution to their co-workers.
- Get management buy-in and sponsorship. Management must really want it, not just go along with it. All levels of management need to support SCM. While implementing SCM, keep a focus on management sponsorship at all times.
- Maintain a sense of humor.
- Be flexible and sensitive to corporate culture.
- Seek out the early success.
- Do not use a critical project as pilot.
- Use a systems approach: Where am I? Where do I want to go? How am I going to get there?
- Success is more likely with lots of preparation, focus on CM and developer needs, breadth of participation, online access to sample process and planning templates, and standard terminology.
- Keep it simple. If it is too complex, or gets in the way, it will not get used.
- Communicate, communicate, communicate.◆

References

1. Software Technology Support Center. "Life Cycle Software Project Management." STSC Seminar, 9 Oct. 2001.
2. Software Program Managers Network. "Little Book of Configuration Management." Software Program Managers Network, Nov. 1998 <www.spmn.com/products.html>.
3. Institute of Electrical and Electronics Engineers. "Standard 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology." New York: IEEE, 1990.
4. Kasse, Tim. Software Configuration Management for Project Leaders. Proc. of Software Technology Conference, Apr. 1997.
5. Berblack, Ronald H. Software Configuration Management. John Wiley & Sons, New York, 1992.

6. Ben-Menachem, Mordechai. Software Configuration Management Guidebook. McGraw-Hill, 1994.
7. Olson, Timothy G., et al. "A Software Process Framework for the SEI Capability Maturity Model: Repeatable Level." CMU/SEI-93-TR-7. Pittsburgh, PA: Software Engineering Institute, 1993.

CROSSTALK did not have room to cover the fundamentals of testing in this month's theme section, "Configuration Management and Test." The Guidelines for Successful Acquisition and Management of Software-Intensive Systems (GSAM) is also a good source for information on the basics of testing and many other software topics. For more information on test, see Chapter 12 of GSAM Ver.. 4.0 at <http://www.stsc.hill.af.mil/resources/tech_docs/gsam4.html>.

About the Author

The Software Technology Support Center (STSC) produced the "Guidelines for Successful Acquisition and Management of Software-Intensive Systems." Visit the STSC Web site at <www.stsc.hill.af.mil/resources/tech_docs> to access all 17 chapters of this document. The STSC is dedicated to helping the Air Force and other U.S. government organizations improve their capability to buy and build software better. The STSC provides hands-on assistance in adopting effective technologies for software-intensive systems. The STSC helps organizations identify, evaluate, and adopt technologies that improve software product quality, production efficiency, and predictability. Technology is used in its broadest sense to include processes, methods, techniques, and tools that enhance human capability. The STSC offers consulting services for software process improvement, software technology adoption, and software technology evaluation, including the Capability Maturity Model® Integration℠, software acquisition, project management, risk management, cost and schedule estimation, configuration management, software measurement, and more.

Software Technology Support Center
6022 Fir AVE BLDG 1238
Hill AFB, UT 84056-5820
Phone: (801) 586-0154
DSN: 586-0154
E-mail: stsc.consulting@hill.af.mil

WEB SITES

Configuration Management Yellow Pages

www.cmcrossroads.com/yp/index.php?oldpage=configuration_management.html

The Configuration Management Yellow Pages is a categorized database of links to configuration management and development resources. The site, originally created by Andre van der Hoek, is now hosted at CM Crossroads in a format that allows any member to add or update a new resource and to review and rate existing ones.

Test and Measurement World

www.tmworld.com

This is the online version of *Test & Measurement World* and *Test & Measurement Europe*, which cover the electronics testing industry, providing how-to information for engineers who test, measure, and inspect electronic devices, components, and systems.

Software Configuration Management

www.sei.cmu.edu/legacy/scm

This is the Software Configuration Management area of the Software Engineering Institute's (SEI) Web site. This area is intended to share the configuration management research done by the SEI between 1988 and 1994 and to provide pointers to other useful sources of information on Software Configuration Management.

Software Testing Institute

www.softwaretestinginstitute.com

The Software Testing Institute (STI) provides industry publications, research, and online services, including a software testing discussion forum, the STI Resource Guide, the Automated Testing Handbook, STI Buyer's Guide, and more.

Data Interchange Standards Association

http://www.disa.org

Home to numerous organizations developing e-business standards, the Data Interchange Standards Association helps individuals and the business community improve business processes, reduce costs, increase productivity, and take advantage of new opportunities.

Software Technology Support Center

www.stsc.hill.af.mil

The Software Technology Support Center is an Air Force organization established to help other U.S. government organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and to accurately predict the cost and schedule of their delivery.

Institute of Electrical and Electronics Engineers

www.ieee.org

The IEEE promotes the engineering process of creating, developing, integrating, sharing, and applying knowledge about electrical and information technologies and sciences. IEEE provides technical publications, conferences, career development assistance, financial services and more.

Practical Software and Systems Measurement

www.psmsc.com

Practical Software and Systems Measurement (PSM): A Foundation for Objective Project Management was developed to meet today's software and system technical and management challenges. The Department of Defense and the U.S. Army sponsor PSM. The goal of the project is to provide project managers with the objective information needed to successfully meet cost, schedule, and technical objectives on programs.

17th Annual Systems and Software Technology Conference Focused on Defense Capabilities

The theme for this year's Systems and Software Technology Conference (SSTC) was "Capabilities: Building, Protecting, and Deploying." Once again the theme targeted the SSTC vision to be the Department of Defense's premier forum to enhance attendees' professional skills and knowledge of systems and software technologies and policies, enabling them to improve the capabilities they provide to the warfighter. This year's conference included more than 180 events to choose from, including general sessions, speaker luncheons, plenary sessions, presentation tracks, and exhibitor tracks.

As the theme suggested, this year's SSTC highlighted the technologies, processes, and practices to deliver enhanced capabilities to the U.S. military. Many of the latest technologies and human ingenuities that make these challenges an opportunity were presented at the 2005 SSTC from April 18-21 in Salt Lake City. Attendees heard from defense and industry leaders, learning from more than 178 expert presentations in topics ranging from acquisition to net centricity to Web service protocols. In addition, they evaluated new and longtime products from more than 210 exhibitors, and networked among the 1,900 attendees focused on similar issues, problems, and solutions.

The SSTC is one of the largest co-sponsored events for U.S. defense-related software technologies, policies, and practices. It is co-sponsored by the U.S. Army, Marine Corps, Navy, Air Force, Defense Information Systems Agency, Department of the Navy, and Utah State University Extension. The co-sponsors have already started planning SSTC 2006, which is scheduled for May 1-4 in Salt Lake City.◆

Gen. John W. Handy, commander, U.S. Transportation Command, and commander, Air Mobility Command, U.S. Air Force, addressed attendees at Thursday's Plenary Session Speaker Brunch about the importance of software and technology in delivering needed supplies to troops in the battlefield.

The Co-Sponsor Panel session included (from left) Donald Reiter, Office of CIO, Department of the Navy; John M. Gilligan, CIO, U.S. Air Force; Maj. Gen. Conrad Ponder, Jr., director of Chief Integration Office G-6, U.S. Army; Rebecca Harris, director, GIG ESE; and moderated by David Mihelcic, DISA CTO.

Photos by Randy Schreifels of the Software Technology Support Center.

Below: To encourage networking among attendees and exhibitors at SSTC, the brunches, refreshment break, and coffee and dessert break were hosted in the trade show among the exhibitors.

Above: An Industry Panel session included (from left) Dr. Jim Kane, president and CEO, Systems and Software Consortium, Inc.; Lou Von Thayer, president, General Dynamics Advanced Information Systems; Dr. David F. McQueeney, chief technology officer, IBM Federal; Grover W. Hall, v.p., Technical Operations, Lockheed Martin Space Systems Co.; Paul Cofoni, president, CSC Federal Sector; and Bob Stow, v.p., Engineering and Technology, BAE SYSTEMS, North America.

Below: Again this year, w
Government's Top 5 Prog
their awards. Articles abo
will appear in September'

Brig. Gen. Robert H. McMahon, director of Maintenance, Ogden Air Logistics Center, Hill Air Force Base, Utah, talks with Ken Wilks of the 309 SMXG, Hill AFB, during the SSTC trade show that featured 210 exhibitors.

winners of the 2004 U.S. grams were presented with ut these winning programs CROSSTALK.

Above: Speaking at the Opening General Session at SSTC were (from left) Maj. Kurt Warner, XVIII Airborne Corps and the 82nd Airborne Division; Priscilla E. Guthrie, chief information officer, Department of Defense; and Brig. Gen. Robert H. McMahon, Ogden Air Logistics Center, Hill Air Force Base, Utah.

Below: Ellen Gottesdiener, EBG Consulting, Inc. was one of 178 presenters at the SSTC during the four-day conference.

A Correlated Strategic Guide for Software Testing

Christopher L. Harlow
The George Washington University Law School

Dr. Santa Falcone
University of New Mexico

Due to the complexity of software development, completed programs are rarely ever flawless. Instead, they exist in a state of constant refinement. As a result, large-scale customized software programs are routinely purchased and employed while still experiencing significant problems. Because of the programs' scale, these problems cost public- and private-sector organizations billions of dollars each year in productivity losses and repair expenses. In the study in this article, test and field data from a large-scale software development project were analyzed using Chi-square goodness-of-fit tests, p-tests, and binary logit regression. The results of this series of calculations support using initial test deficiencies in mid-production software tests to guide later iterative testing to increase deficiency exposure. When used during software development, this strategic test guide is expected to improve testing, expose problems earlier, help lead to higher quality end-products, and ultimately reduce the large losses organizations experience due to customized software defects.

Information technology advances have equipped the U.S. military with an extraordinary arsenal of weaponry and supporting materiel. To keep the arsenal current, the Department of Defense (DoD) modified its acquisition strategy to reflect the iterative nature of information technology (IT) development. Both public- and private-sector large-scale software-intensive acquisitions use iterative strategies to conduct the *try-before-you-buy* testing of these purchases. This article briefly reviews literature on software testing and describes an iterative test strategy used in an actual large-scale military software acquisition.

The Timing of Software Testing

In the past, a series of increasingly stringent development tests controlled quality and guided DoD large-scale customized IT acquisitions through the procurement process. Right before acquisition fielding, an operationally realistic test evaluated the product's functional effectiveness within its intended environment [1]. Thus, multimillion-dollar purchase decisions depended largely upon successful operational tests at the end of production.

In the United States in 2002, the National Institute of Standards and Technology calculated the annual cost of these operational test failures in the U.S. public and private sectors at $59.5 billion [2]. An independent study found that more than half of IT acquisitions doubled their initial budget and schedule projections, the average acquisition provided only 61 percent of the desired functionality, and one-third of software-intensive projects were ultimately cancelled [3].

In IT acquisitions, the desired software end-product is seldom clearly defined [4]. The identification of requirements and the development of the software to fulfill them are progressive and often become intertwined, fueling a spiral in which unanticipated requirements emerge [5]. Government agencies have referred to this process as *evolutionary acquisition* and *spiral development*. Their civilian counterparts use the phrases *agile development* and *extreme programming* [6].

In the recent past, government testing was insensitive to this process of software development. In 2000, the U.S. General Accounting Office (GAO) harshly criticized DoD evaluation methodologies and concluded that late operational test failures consistently plagued DoD purchases, particularly software-intensive systems [7]. Accordingly, the GAO denounced the traditional practice of employing testing as a *watershed event* in sole support of fielding decisions. The failure of traditional practices combined with the technological revolution led directly to the development of alternative test strategies, including the method described in this article.

After studying government and private practices, the GAO concluded that the most effective method of exposing deficiencies was iterative operational testing. GAO further recommended the philosophy developed by AT&T: *Break it big early* [7]. According to AT&T, the earlier a problem is discovered, the easier and less expensive it is to fix, making software development more cost-effective. Other benefits include encouraging technological exploration and advancement, controlling risk [8], exposing unanticipated requirements, and enabling their eventual fulfillment.

Spurred by success, incremental testing has become the industry standard for software development projects. For example, during the acquisition of the Theater Battle Management Core System, evaluators successfully mitigated risk by operationally testing each basic system component prior to fielding. The test team also observed that the incremental strategy facilitated requirements development as operators became increasingly familiar with the system [9]. The DoD employed the strategy with several programs, significantly improving its acquisition and testing practices.

In addition to incremental testing, there is also near unanimous agreement about using operationally realistic testing early in the development cycle. Past and present DoD Operational Test Agency leaders assert that early operational tests streamline the production process, improve requirements definition, and provide valuable feedback [10, 11]. The Global Command and Control System test team, one of the first DoD programs to effectively use it, found that early operational testing increased operator system familiarity, which, in turn, increased the number of deficiencies exposed during testing [12].

In March 2002, the Giga Information Group estimated that two-thirds of private-sector software projects would employ *agile development* within the next two years. Using terms nearly identical to those in the government lexicon, the group identified these projects as those that are divided into smaller phases and require more frequent testing [6]. According to the prevailing opinion in

the public and private sectors, the best response to the challenges of IT testing is incremental operationally oriented evaluations beginning in the early stages of system development.

Early and Continuous Software Testing

While the community has unanimously embraced the need for earlier testing, there is no consensus on how to conduct early testing. Experts have proposed merging developmental and operational tests, instituting sustained independent operational testing, using Bayesian hierarchical models [13], and replacing operational testing with mathematical models. In addition, some argue that those who will actually operate the completed system should be active observers during early developmental tests [3].

Ideally, involving primary users in early testing expedites identification, prioritization, and resolution of exposed deficiencies, and results in lower costs due to reduced operational testing [4]. This modified early testing with primary users still culminates in a final operational test. Thus, while using operators as testers and separating large-scale software acquisitions into incremental segments conform to the new paradigm, this modified testing also retains the traditional concept of a final operational test of the whole system.

Several experts eschew the implicit requirement for physical testing and suggest the utilization of risk assessment models. Thompson proposes four strategies ranging from the traditional production model to various levels of operationally realistic testing, determined by a variety of statistical inputs [14]. Expanding on Thompson's proposal, Arnold's mathematical model would effectively eliminate the need for detailed testing of specific aspects of a potential acquisition, ostensibly saving substantial resources [15]. While the substitution of mathematical modeling for testing is not entirely accepted within the community, proponents of mathematical modeling assert that accurate models can be built. Systematic refinement of mathematical models is anticipated to improve their predictive abilities [16].

The model or strategy proposed in this article employs Thompson's idea of using statistical data in testing. However, the use of statistical data here will identify specific test activities, not just test levels. It also will provide a guide for the iterative incremental testing process rather than an outright replacement of test activities.

Testing Using Correlation

The DoD commissioned a study in May 2000 to evaluate the accuracy of operational tests by comparing test results of 11 systems to wartime field data [17]. The limitations of this study were as follows: Low-level task-based test data was compared with strategic-level operational data; a standardized deficiency tracking method before and after fielding was not used; all 11 systems had been significantly altered after testing and many of the modifications were not documented; and, finally, there was an extremely high turnover rate of test personnel. Turnover is not isolated to the military: Civilian test teams often encounter turnover rates as high as 80 percent in test periods only a few months in length [18]. To credibly compare data from one iterative test to another, the identification of test and field deficiencies should be standardized and the only substantive system changes should be fixes to deficiencies exposed during official testing.

The closest approximation to this article's correlation model was the incremental development and testing of IBM's integrated results database for the 2000 Summer Olympics. IBM utilized standard Orthogonal Defect Classification to categorize system defects along an x-y graph, revealing defect similarities and trends the IBM team termed *triggers*. The IBM team assumed that a high trigger incidence rate revealed an area of weak code and "used the insight to guide test teams toward more effective defect discovery" [18]. The IBM trigger metrics represent an initial attempt by the software industry to relate test data to future performance in a manner aimed at improving the production process.

As noted earlier, the specific requirements of large-scale software programs are unclear at the inception of the design and production process. The correlation of iterative test results is proposed here to enable discovery, more specific clarification, and fulfill end-user needs during production and testing. This approach also helps manage the eight factors that influence success or failure during software testing: production, infrastructure, training, personnel, communications, operations, maintenance, and environment [1].

Regarding the first factor, production, according to software production expert M.S. Phadke, faulty coding tends to be regional and hence predictable [19]. Regional, as it is used here, refers to either a section of the code or a physical location where the code is generated. As per Phadke's observation, there will be regions of computer code that are highly correlated with deficiencies in the initial stages of design and production (prior to the first test). These regions should be more closely observed in the first test for the specific types of deficiencies identified during production. Any region with a large number of deficiencies during the first test would then be targeted for emphasis in the next production cycle and for extensive testing during the second test. Comparing by region the deficiencies exposed during the first test to those exposed during the second test would again target the regions that should receive more emphasis in the following production cycle and extensive testing during the third test, and so on, iteratively encouraging the optimal identification of unexposed deficiencies.

Comparing this approach to IBM's, the triggers identify only the regions of code with problems. They do not indicate the relative significance of the difference between the regions and do not track the eight factors. In this approach, data about the eight factors was collected during two tests of a large-scale software program. This enabled analysis of the deficiencies exposed and each of the eight factors. This analysis identified the regions of code that were problematic, the difference between regions, and which of the eight factors were significantly related to the exposed deficiencies.

Methodology

The data for this study came from two tests (TEST1 and FIELD) of the large-scale software program Deliberate and Crisis Action Planning and Execution System (DCAPES), designed to serve an information technology need of the U.S. Air Force [20]. For TEST1, operator information and the results of the testing of 53 tasks within the first increment of DCAPES were collected from five worldwide locations during two weeks in June of 2001. The 53 specific tasks were organized into the following six categories: operations, logistics, information analysis, reference file access, manpower data management, and system administration [20]. Fifty-two operators assessed the system's capabilities and performance in 4,592 discrete actions in a simulated operational environment. Data collected in TEST1 included the overall recording of the outcomes of the test, deficiency documentation, operator information,

and operator satisfaction surveys.

The first increment of DCAPES was then fielded in March 2002. FIELD, the second data collection effort, involved recording deficiencies that were found in the normal operation of the first increment of DCAPES after it was fielded. Normal operations consisted of an estimated 225,000 discrete actions by approximately 2,000 authorized operators in 16 worldwide locations over a 10-month period. Data collected in FIELD included deficiency documentation and operator information.

Performance failures or deficiencies were documented using the exact same procedures and codes for both TEST1 and FIELD. Operators documented deficiencies with standardized forms extensively describing the problem and the circumstances leading to its exposure. While documenting the problem, the operator assigned a priority describing the impact of the deficiency on the organization's mission on a five-point numeric scale. A priority one deficiency described a catastrophic mission failure, while a priority five described an easily circumvented minor inconvenience.

After the operator assigned an initial priority, a Deficiency Review Board (DRB) convened to review the legitimacy of the problem and the appropriateness of the assigned priority. The DRB consisted of representatives from the software developer company, the operating community, and an independent government evaluator as chairman ensuring impartial review. TEST1 and FIELD both followed these same procedures, utilizing identical definitions and DRB members. This replication ensured comparable treatment of test and field deficiencies. The DRB also rejected duplicate deficiencies, ensuring each problem was documented only once. After the DRB assigned a final priority, the problem was officially documented in another database that consolidated and segregated test and field deficiencies, making this information readily available to authorized personnel.

A series of χ^2 goodness-of-fit tests, p-tests, and binary logistic regression equations were conducted on the data collected to answer the problem statement, *"Can the correlation of software test and field data be used to guide testing to increase deficiency exposure?"* The analyses are organized and discussed in three sections: system functionality, organizational trends, and operator data. System functionality refers to the performance of the DCAPES system. Organizational trends refer to how organizations impact the performance of the DCAPES system. Operator data refers to how operators impact the performance of the DCAPES system.

Findings

Regarding system functionality, this study began with the assumption that coding errors tend to be regional. One implication of this assumption is that defects are interrelated and may be predictable. Analysis of the results of the testing of the 53 system tasks within the six functional categories supports this assumption. The data indicates that tasks and categories with high execution quantities had more field deficiencies. These tasks and categories were more complex, containing a broader range of functions made possible through additional lines of code. Due to this complexity, these areas were more susceptible to errors. The binary logistic regression results also indicate a significant relationship between the execution volume of complex code and incidence of errors. Applying this finding, a test director could increase the testing of regions with high volume execution.

Another affirmation of the regionality of errors was evident in that regions of code plagued by failures during the first test exhibited additional defects in the field test. The time and resource constraints placed on testing prohibit exposure of every deficiency in any single round of testing. As a result, it is not surprising that defects remain undiscovered even in areas well tested in previous iterations. Again, a test director could alter test activities to emphasize regions with substantial failures in past tests.

An organization-wide trend that had significant impact was the adequacy of training provided to operators. The findings indicate that the categories in which operators had inadequate training had higher field deficiency quantities. Intuitively, this results from the inability of operators to stringently test these categories in the first test. This study found that DCAPES training problems were more related to the categories of tasks than the geographic location of the task execution. This provides valuable information because training courses are typically organized along both lines with category specialists trained as a group at each location.

To improve a test in progress before its completion, the awareness that operators have inadequate training in specific categories should be used to redirect testing to emphasize the reported categories using only well-trained operators. However, in an incremental development project, this feedback concerning categories in which operators are reporting significant training problems could be provided to instructors and the training adjusted to prevent inadequate training from negatively impacting later tests.

The operator data identified significant factors to guide the testing process and factors that are not as important as conventional wisdom suggests. For example, prevailing opinion is that defect exposure detracts from test execution. A variety of calculations revealed that the expected negative relationship between execution and deficiency submission did not occur. Instead, the significant relationship in this regard was that operators who submitted false deficiencies (false deficiencies are those withdrawn by the submitter or rejected by the DRB) exposed significantly fewer legitimate defects overall. This calls into question the quality of testing accomplished by these operators and suggests the participation of operators who submit false deficiencies in past tests should be curtailed in future tests.

Current conventional wisdom also asserts that test planning should entail the meticulous creation of a simulated environment that closely approximates the operational setting. The findings of this study, however, indicate that some facets of this environment may not be necessary to simulate so meticulously. For example, significant resources are devoted to recruiting representative operators; however, the findings here were that field deficiency exposure rates increase with operator skill level and system experience. Using operators in the testing process with great skill level and system experience apparently would be a more effective and efficient way to expose deficiencies than ensuring that operators representing the full available range of skill levels participate in the test.

Finally, tasks with lower operator satisfaction levels were found to contain unexposed deficiencies. These operator satisfaction results provide information about system performance that can be used to guide testing. Within the time period of one test, operators often rotate in and out. Using this to advantage, a test director could steer subsequent testing toward areas receiving lower marks on satisfaction ratings of operators rotating out. In a spiral development effort, test directors could also schedule later tests to emphasize areas receiving lower operator satisfaction ratings in previous iterations.

Conclusion

The statistical analysis within this study has specific application to the remaining tests in the development of DCAPES and the development of similar large-scale systems. Through publishing this article, information about the approach suggested here will be incorporated into the body of information about software testing. Then, as the methods here are more widely employed, they will refine testing and help improve end-product quality.

As mentioned earlier, the National Institute of Standards and Technology estimates software defects cost the United States nearly $60 billion annually. The Institute also estimates that practical approaches to software development and testing could reduce this figure by 37 percent [2]. Essentially, $20 billion can be saved each year through the implementation of relatively simple software production models. The correlation model proposed in this article is an initial attempt at capturing a small percentage of these funds. While not the most statistically elegant method, it is a method that most professionals in the work world will be able to understand and, most importantly, use. Public- and private-sector large-scale software development projects may both realize tremendous gains from a statistical correlation model.

The data analysis referred to in this article is available upon request.◆

References

1. McGowen, D.J. "C4I Operational Test and Evaluation in Support of Evolutionary Acquisition Strategy." ITEA Journal of Test and Evaluation 21.2 (2000): 34-39.
2. RTI. "Planning Report 02-3: The Economic Impacts of Inadequate Infrastructure for Software Testing." Washington, D.C.: National Institute of Standards and Technology, May 2002 <www.nist.gov/director/prog-ofc/report 02-3.pdf>.
3. Maybury, M., A. King, and J. Brooks. "Software Intensive System Acquisition – Best Practices." 2003 Acquisition Conference, 28-30 Jan. 2003, Arlington, VA <www.sei.cmu.edu/products/events/acquisition/2003-presentations/maybury.pdf>.
4. Assi, S., and M. Thompson. "Alternative Test and Evaluation Strategies for Command and Control Systems." ITEA Journal of Test and Evaluation 20.2 (1999): 21-25.
5. Axiotis, G. "Evolutionary Acquisition and Operational Testing, Time for a New Approach." Evolution 99 (1999): 1-5.
6. Sliwa, C. "Users Warm Up to Agile Programming." Computerworld 18 Mar. 2002 <www.computerworld.com>.
7. General Accounting Office. "Best Practices: A More Constructive Test Approach Is Key to Better Weapon System Outcomes." Washington: GAO, July 2000.
8. Cast, M. "Military Links Developmental and Operational Testing to Meet Technology Challenges of the 21st Century." Program Manager July-Aug. 2000: 16-18.
9. Zyroll, T., A. Johnson, and B. Connally. "Government Testing Philosophy Redefined for Theater Battle Management Core Systems." ITEA Journal of Test and Evaluation 19.1: 25-47.
10. Whittmeyer, J. "Meet DoD's Top Advisor on Operational Test and Evaluation." Program Manager May-June, 1996: 2-8.
11. Besal, R.E., and S.K. Whitehead. "Operational Testing: Redefining Industry Role." National Defense Magazine Sept. 2001 <www.nationaldefensemagazine.org/archives.htm>.
12. Bailey, M., and M. Spencer. "Engineering the Largest C4I Operational Test in Naval History." ITEA Journal of Test and Evaluation 20.1 (1999): 26-33.
13. Graves, T. "Hierarchical Models for Software Testing and Reliability Modeling." 2003 Quality and Productivity Research Conference, IBM T.J. Watson Research Ctr., Yorktown Heights, N.Y., May 21-23, 2003.
14. Thompson, M., and E. Montagne. "Using Risk Assessments to Determine the Scope of Operational Tests for Software-Intensive System Increments." ITEA Journal of Test and Evaluation 19.1 (Jan. 1988): 42-47.
15. Arnold, A.G., and W.F. Kujawa. "Test and Evaluation of Complex System of Systems." ITEA Journal of Test and Evaluation 20.3 (Mar. 1999): 33-36.
16. O'Bryon, J.F. "Meet MASTER-Modeling and Simulation Test and Evaluation Reform. ITEA Journal of Test and Evaluation 20.4 (Apr.1999).
17. Brown, S.O., and K.E. Murphy. "Air War Over Bosnia: OT&E Impact on USAF Systems Used Over Serbia." Kirtland Air Force Base, NM: Air Force Operational and Test Evaluation Center, 2000.
18. Bassin, K, and S. Biyani.. "Metrics to Evaluate Vendor-Developed Software-Based on Test Case Execution Results." IBM Systems Journal 41.1 (2002): 13-30 <www.research.ibm.com/journal>.
19. Phadke, M.S. "Planning Efficient Software Tests." CROSSTALK Oct. 1997 <www.stsc.hill.af.mil/crosstalk/1997/10/index.html>.
20. Harlow, C. "DCAPES Operational Assessment Final Report." Kirtland Air Force Base, NM: Air Force Operational and Test Evaluation Center, 2001.

About the Authors

Christopher L. Harlow currently attends The George Washington University Law School. Previously, he was a test manager for three years at the Air Force Operational Test and Evaluation Center in New Mexico where he designed and executed worldwide operational tests of The Deliberate Crisis Action Planning and Execution System. Harlow has a Bachelor of Science in economics from the U.S. Air Force Academy and a Master of Public Administration from the University of New Mexico.

The George Washington University Law School
E-mail: charlow@law.gwu.edu

Santa Falcone, Ph.D., is an associate professor in the School of Public Administration at the University of New Mexico and teaches public budgeting and public finance.

University of New Mexico
2085 Anderson School of Management
Albuquerque, NM 87131
Phone: (505) 277-4934
Fax: (505) 277-7108
E-mail: falcone@unm.edu

"But the Auditor Said We Need to ..." Striking a Balance Between Controls and Productivity

Greg Deller
Capgemini Government Solutions

This article discusses the common gap between audit-recommended controls and those typically implemented by software project teams. The article lists commonly misunderstood audit recommendations and provides a clear explanation of what the auditors are really seeking from software project teams.

All too frequently audit recommendations are received by software project teams and immediately hung up on a dartboard for target practice. One reason they are not well received is because they are misunderstood. Audit recommendations are simply designed to be guidelines to achieve *reasonable* control, not specific instructions that hinder an organization's ability to be productive and efficient.

With the appropriate level of communication between auditors and project teams, audit recommendations can actually be extremely helpful for organizations to reduce the risk of excessive defects, delayed releases, cost overruns, and unmet customer requirements.

Unfortunately, poor communication between auditors and project teams is one of the main reasons that organizations are not able to effectively reduce their systems-based risk exposure. Frequently this poor communication causes one of two scenarios. The first is that project teams overstate the effort necessary to become compliant with audit recommendations, thinking that what the auditors are asking is entirely unfeasible. In the second scenario, project teams underestimate what the auditors are recommending because they missed the intent of the recommendation.

This miscommunication between auditors and software project teams can be thought of as *recommendation gap*. This article lists the most frequent scenarios of recommendation gap in an effort to shrink the gap. By analyzing the *intent* of the audit recommendations as opposed to their specific wording, much insight can be gained to reduce the risk of a failed project. Although this article is directed toward software projects, the viewpoints can apply to a variety of system initiatives.

Classic scenarios of overestimating the effort necessary to comply with recommendations are presented below; following these are scenarios of underestimating the effort necessary to comply with recommendations.

Classic Scenarios of Overestimating Effort

Project Plan

A very common audit finding is the lack of a project plan. Auditors recommend developing a project plan (as opposed to just a project schedule) for new software projects that are important to an organization (i.e., financially, politically, or strategically). The purpose of the project plan is to communicate to others how the project activities will be controlled.

Many project managers perceive project plans as oversized documents that no one has time to develop or read. On the contrary, they cannot afford to go without a project plan; the plan can communicate necessary information to the project team instead of project managers communicating the information multiple times or not communicating the information at all, thus creating chaos. It should also be noted that an indirect benefit to creating and maintaining a project plan is that it forces project management to think about critical topics otherwise not considered.

Auditors are not looking for the project plan to meet a certain length or depth requirement; they are just concerned that pertinent components are addressed (see Table 1 for potential components). For instance, depending on the size of a project it is conceivable that topics such as stakeholder identification and interaction can be addressed in a paragraph.

Table 1: *Recommended Project Plan Components*

Recommended Project Plan Components	
System Life-Cycle Considerations	Milestones
Technical and Management Tasks	Data Management
Budgets and Schedules	Risk Identification
Resource and Skill Requirements	Stakeholder Identification and Interaction

System Life-Cycle Methodology

Another very common audit recommendation is for organizations to adopt and implement a formal system life-cycle methodology. Organizations tend to take the extreme of this recommendation by either adopting a robust yet unwieldy system life-cycle methodology that virtually no one is trained on, or ignoring the recommendation altogether, seeing it as a mountainous task.

This recommendation can be addressed with relative ease by a project team considering its strengths and weaknesses and adopting a basic life-cycle methodology that exploits their strengths and overcomes their weaknesses. There are a number of proven life-cycle methodologies that are publicly available; creating one from scratch is generally unnecessary. Widely used methodologies include waterfall, rapid prototyping, incremental build, multiple build, spiral, fast-track, and hybrid.

Project teams should select a methodology that can be easily implemented and will not require extensive training. This leads to an area that most organizations fall short on – training and implementing the methodology. Auditors are not concerned that a methodology that fills five binders has been developed. Instead, they want to see that a valid methodology appropriate for the specific project team has been adopted and implemented.

Change Management

Change management policies and procedures do not need to handcuff an organization's systems' staff, but they should be commensurate with the risks of the specific environment. Modifications to a missile control system present a higher risk than modifications to a video library system; therefore, change management policies and procedures must be comparatively more structured. Auditors simply want to see that a change management process is carefully designed for the specific operational and technology environment as well as being implemented entity-wide.

An important component of this

includes the organization's ability to ensure "consistency in the management and control of software changes" [1]. This begins with establishing a policy and procedure describing how program changes are to be made. These policies and procedures should incorporate a structured process to ensure that system changes are requested, tested, and approved prior to implementation, as opposed to an informal approach (such as ad-hoc troubleshooting).

While a structured process does add new requirements, it can be implemented without requiring a significant increase in resources, if designed appropriately.

> Increasingly, organizations are recognizing that effective change control management is a key to assuring that the product delivered is indeed the product intended and expected. [1]

If the product is delivered as intended and expected the first time, fewer resources are needed in the long run to fix the problem.

Disaster Recovery

Disaster recovery considerations should be addressed from the inception of a project. Should the Internal Revenue Service cut over to an Internet-based tax filing system without considering high availability? Should the Navy implement a new fleet maintenance tracking system without considering a back-up scheme? The answers are obvious.

On the other end of the spectrum, does a small construction company need to consider disaster recovery for their Internet access? They do if they are implementing a new payroll system utilizing a third-party payroll processor that can only receive payroll submissions via a file transfer from a specific Internet Protocol address.

All of these scenarios present a risk of continued operations that should be discussed among the project team and stakeholders prior to the implementation date. However, if the topic is not discussed during the early stages of the project, implementation decisions may be made about the system that negatively affect disaster recovery capabilities. Disaster scenarios and contingencies should at least be discussed and documented among all key personnel when developing a new system. This can be as simple as brainstorming about the risks, mitigating circumstances, and contingencies during a team meeting at key milestones during the project.

Audit requirements should not dictate across the board that systems are fully redundant and disaster recovery plans meet the five-binder requirement. What they do require is that disaster risks are identified and the impact of the risks is assessed. Based on the risk analysis and business impact analysis, management can then determine what level of risk they are willing to accept versus the cost. Reasonable contingencies should be identified and implemented to reduce the overall risk exposure. Again, these planning exercises may be as simple as getting the appropriate personnel in a room to discuss and document the decisions.

Further, there is a common misconception within organizations regarding who should be responsible for addressing business continuity and disaster recovery. The key is not who is responsible for it or who manages the effort, but who provides input to the disaster recovery planning effort. Input to the disaster recovery effort must come from all stakeholders including management, end users, and systems personnel. It should be a coordinated effort where business and technical issues are discussed.

"Unfortunately, poor communication between auditors and project teams is one of the main reasons that organizations are not able to effectively reduce their systems-based risk exposure."

Data Conversion

Auditors typically recommend a structured approach be taken with most complex project tasks. Data conversion is no exception. The *recommendation gap* in this area is generally more prevalent for small- and medium-sized projects than with large projects. Large projects generally identify a data conversion approach and perform data conversion activities that support the approach. These data conversion activities can include development of the following documentation:

- List of all of the legacy data that must be cleansed and loaded into the new system.
- Mapping diagrams for data from the legacy system to the new system.
- Conversion plan, which includes the approach (e.g., manual entry, file load, transaction load, automated program, etc.).
- Data conversion reconciliation and balancing procedures.
- Error resolution for data conversion errors.
- Post-migration review and approval from an appropriate stakeholder.

Although these items take a significant amount of effort to prepare for a large project, they can be prepared for small- and medium-sized projects with limited resources by focusing on the intent of the documents. The intent is to develop an effective and controlled approach to data conversion. An important element of a controlled approach is that it be well formulated and communicated to all involved, thus the need for documentation (e.g., plans, mapping diagrams, error resolution, etc.).

Again, there is no requirement regarding the length or depth of the data conversion documentation, only that it addresses the key decisions (e.g., conversion method, categories of data converted) and documents the reconciliation process so it can be re-performed.

End-User Testing

End-user testing can go a long way toward developing a relationship with customers and gaining their support. But primarily end-user testing is designed to catch defects by using a tester that may be more familiar with the subject matter and provide a fresh set of eyes.

> User acceptance testing identifies defects before they get into production and gives the user community a chance to *kick the tires* on the system before it goes live. [2]

Why wait until a system is already implemented to learn that the menu names do not meet user needs, or that a system formula for a calculation is incorrect?

End-user testing can be as simple as having a sample of end users identify their key activities and execute them in the test system, or by having end users review system-generated reports for validation of accuracy and effectiveness. The key to end-user testing is user confirmation of the accuracy and functionality of the system. This can be easily accomplished by working with the users to list the key activities that they perform (i.e., test items) and

	Security Role (User Profile)			
Functions	Clerk	Accountant	Supervisor	Controller
Journal Entry (JE1)	✓	✓	✓	
Journal Inquiry (JI1)	✓	✓	✓	✓
Journal Post (JP1)		✓	✓	
Journal Reporting (JR1)	✓	✓	✓	✓

Note: The screen name is listed in parentheses.

Table 2: *Security Matrix*

request that they test the key activities in the system for accuracy and effectiveness.

During the end users' test process they should document their validation of each of the key processes listed and clearly document any errors or problems. Not only does the list help the users know what to test, but it indirectly helps to ensure that they perform each of the components of the test. Auditors' primary concern is that end users perform key process testing and confirm the accuracy and functionality prior to implementing a system, not that the end users test every intricate element of a system.

Go-Live Approval

End-user management and project team management must perform an exercise to identify criteria that must be satisfied prior to implementation of a change into the live environment. They must then review compliance with the criteria and collectively approve a system's readiness prior to implementation. However, this may be as simple as gathering the appropriate people in a meeting to gain their documented approval that go-live criteria have been satisfied.

Auditors are not as concerned about the details of the go-live review as they are about the process of evaluation. This includes how the go-live criteria were identified and the process to review and approve compliance with the go-live criteria. Accountability and ownership for the go-live decision can be provided by end-user management and project team management signing the go-live criteria checklist and maintaining it with the central project files.

Classic Scenarios of Underestimating Effort

The discussion will now switch to scenarios where project teams frequently underestimate the level of effort required to meet audit recommendations. Again, the cause of this recommendation gap is attributed to miscommunication regarding the *intent* of the auditors' recommendations.

Application Security

Project teams have a tendency to be focused on ensuring that the requested system functionality is implemented within, generally, a tight timeframe. Often, that leaves application security as merely an afterthought or a task that is quickly addressed by a small team, not consisting of appropriate personnel, based primarily on assumptions of needed user access.

Instead of addressing security at the tail-end of the project, auditors recommend that security be addressed at each phase of the project, including project initiation and planning. From a logical perspective, this entails appropriate personnel identifying sensitive data and function access, establishing roles (i.e., user profiles) for users, and then assigning data and function access to the roles. From a physical perspective, application security design must be performed simultaneously and integrated with the system design process to help eliminate downstream conflicts.

An accounting-based example of this conflict exists when a user needs access to *Journal Inquiry* on a screen, but should be prevented from accessing *Journal Entry* on the same screen (assuming security is restricted at the screen level). A possible reason for this conflict is that system designers did not consider security considerations when they were designing the system so it did not occur to them that the two incompatible functions should be on different screens.

Logical security specifications can be documented in a security matrix (for example, refer to Table 2). In a security matrix, groupings of users that have the same security requirements are formed into security roles. Then specific functions in the application are assigned to the security roles. Depending on the implementation, the security matrix can even be used to document the mapping from the logical design to the physical design if the function column is granular enough to match physical application security (e.g., screen names).

Independent Migration

Although audit recommendations repeatedly suggest that all code changes be migrated to the live (production) environment by an independent person, that alone is not sufficient. The intent of the independent migration is so that an independent person can review the change and ensure that it has been through proper testing, documentation, and review prior to implementation. However, limited benefit is gained by having an independent person migrate the change to the live environment if he or she does not perform a review or provide oversight for the change. This independent review applies to any program changes to the live environment, regardless of the source of the change (e.g., development/test environment).

Varying levels of review can be performed by this independent person, but the goal is for the independent person to ensure the appropriateness of the change. As an example, the independent reviewer could verify that a Change Control Board has authorized the change, or the independent reviewer could be responsible for performing his or her own review and test of the code. Then the independent reviewer is responsible for raising the issue through proper channels if the change is not appropriate.

Risk Management Plan

Although risk management planning is gaining attention with large projects, it is still rarely addressed in small- or medium-size projects. However, even small projects developed in the current economy have management visibility that justifies the need for risk management.

> The first objective in a risk management plan is to prevent undesirable situations from occurring. The second objective is to reduce any negative consequences when something undesirable does occur. [3]

For example, an organization may address the first objective by aggressively compensating employees who are the sole provider of a skill to prevent their departure. Then, they may address the second objective by cross-training employees to reduce the negative consequence if the resource does leave the organization. Note that disaster recovery planning meets the second objective of risk management because it aims to reduce any negative consequences when something undesirable does occur.

As Dr. Richard Bechtold indicates, a plan should be developed that addresses how the project team does the following:

> … intends to identify, evaluate, prioritize, mitigate, and manage project risks. Select the top five or 10 risks to be the primary focus of your risk management activities

and describe each. Document the probability and impact of each risk and calculate the resulting risk exposure. [3]

This traditionally has been a topic that is addressed informally by project managers that believe they can see problems coming on the horizon and perceive that their projects are small enough to have visibility to all of the risks. Instead, a proactive approach to risk management is typically recommended by auditors even for small- and medium-size projects. The designation of resources to perform risk management activities must be performed during project planning to avoid the scenario of risk management activities becoming a drain on the project team's resources.

Closing the Recommendation Gap

This article has mentioned the recommendation gap several times to convey the prevalence of poor communication between auditors and project teams regarding audit recommendations. While there is no doubt that both sides are responsible for fostering open and honest communication, the auditors are primarily responsible for ensuring that recommendations are clearly communicated and there is no confusion about the intent of the recommendations. The following list provides actions that can be performed to close the recommendation gap:

- Project teams can request to have daily or weekly briefings on issues or audit concerns that arise during the audit. Also, the two parties should meet at the end of an audit to verbally discuss all recommendations prior to a report being finalized.
- Auditors can provide the project teams with examples and templates of documents that they recommend the project teams develop.
- Auditors can brief line management on the details of the recommendations prior to briefing upper management since the line managers will typically be directly responsible for addressing the recommendations.
- Project teams can request clarification (verbal or written) on audit reports prior to devising responses or action items.
- Auditors can provide the source for their recommendation such as the Software Engineering Institute's Capability Maturity Model®, Project Management Institute standards, Control Objectives for Information and Related Technology Audit Guidelines, Federal Information Processing Standards publications, American Institute of Certified Public Accountants standards, and Financial Accounting Standards Board (or other regulators) standards.

Summary

There is a common gap between audit-recommended controls and those typically implemented by software project teams. Often, that recommendation gap can be attributed to miscommunication between the auditors and those being audited. Miscommunication results from the project team either overestimating or underestimating the effort needed to comply with audit recommendations. Extensive communication between the auditors and project teams is necessary to close the recommendation gap. Experience shows that the implementation of audit recommendations will reduce risk and can also lead to improved efficiency and effectiveness.◆

References

1. Vallabhaneni, S. Rao. Certified Information Systems Auditor Examination Textbook Vol. 1: Theory. 2nd ed. Los Angeles, CA: SRV Professional Publications, 1996.
2. Mogyorodi, Gary E. "Let's Play 20 Questions: Tell Me About Your Organization's Quality Assurance and Testing." CROSSTALK. Mar 2003: 30.
3. Bechtold Ph.D., Richard. Essentials of Software Project Management. Vienna, VA: Management Concepts, 1999: 110.

About the Author

Greg Deller is employed by Capgemini Government Solutions. Previously, he was a manager with KPMG's Information Risk Management practice. Deller is a Certified Information Systems Auditor with more than eight years of systems-based risk management and consulting experience. He has advised over 110 clients on the security and controls in their information systems environment. Deller has a Master of Science in information systems from George Mason University in Fairfax, Va.

Phone: (703) 244-5202
Fax: (208) 723-1537
E-mail: dellerg@yahoo.com

Finding CM in CMMI

Anne Mette Jonassen Hass
DELTA

Configuration management (CM) acts as a central nervous system in system development, and is a prominent key process area and generic practice in Capability Maturity Model® Integration (CMMI®) and other maturity models. As such, implementing and improving CM in the company may seem like an overwhelming task. This article takes you through the requirements in CMMI step by step and offers practical and ready-to-use advice on how to get started and how to get better in this interesting and rewarding area.

Configuration management (CM) acts as a central nervous system in system and software development. If you do not have it, you definitely need it. When you implement it, do it once and very, very carefully. When done, you cannot imagine a life before it.

Every company developing products should consider CM. It is integral to process improvement and the concept of using maturity models to support process improvement, which is becoming more and more integrated in the industry. In the Software Engineering Institute's Capability Maturity Model® (CMM®) Integration℠ (CMMI®), CM plays a prominent part as a key process area at CMMI Level 2 in the staged representation. In both the staged and the continuous representation, CM is a generic goal for all key processes.

Staged representation is similar to CMM V.1.1 where each key process area is assigned to one specific maturity level out of the five defined (initial, managed, defined, quantitatively managed, and optimizing). In continuous representation, each key process area can reach the six defined capability levels (incomplete, performed, managed, defined, quantitatively managed, and optimizing).

CM is, however, not something that most companies are comfortable using. The author has participated in more than 50 assessments in Denmark over the last eight years. In each assessment, a list of the top five candidates for process improvement was produced. The three most frequently appearing processes follow:

1. Project Management (75 percent).
2. CM (55 percent).
3. Test (51 percent).

More than half the companies needed to improve their practices for CM. This made it the second most appearing process.

Introducing CM into a company is, however, not an easy task. Not many controlled experiments on process improvement with the subject of CM exist. For example, under the project Software Improvement Case Studies Initiative (SISSI)[1] – The Business Benefits of Software Best Practice," which has been performed for the European Software Institute, only five out of 50 experiments had CM as their subject [1, 2, 3, 4, 5]. Some of the trends in the conclusions of these reports follow:

> *"The success or failure of a company introducing CM may depend on the people allocated to the task. The person driving the initiative must be a fiery soul or a true believer."*

- Introduction of CM is a great advantage for the company.
- Management support is essential.
- It is important to perform pilot tests of the CM processes before they are rolled out at a greater scale.
- Introduction of CM is difficult.

When a company is facing the task of improving its CM, it may seem like an overwhelming task. Where do you start? Where do you go? How do you get there? Using CMMI for example, a company can get an appraisal of the capability of their CM process. But even without a formal assessment, the definition of the CM Key Process Area may be used as a guideline for implementation and improvement of CM in a company.

CM Benefits

CM brings facts and control to project management. Now, what is that worth? It is hard to say – not many have estimates of the unforeseen problems that come from lack of control and unknown facts.

The savings from a working CM system may stem from preventing one or more of the following problems (among others):

- Work is based on the wrong basis.
- Errors once corrected reappear again.
- Unwanted functionality is introduced, or functionality is forgotten.
- It is difficult or impossible to reestablish a running product.
- It is difficult or impossible to make changes in an existing system.
- It is difficult or impossible to revert to an earlier version that works.
- It is difficult to establish stable decisions because decisions are constantly reversed.

CMMI Definition

The CMMI for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing V.1.1 states the following:

> The purpose of CM is to establish and maintain the integrity of work products using configuration identification, configuration control, configuration status accounting, and configuration audits. [6]

In CMMI, a number of goals are defined for each key process area and a number of practices are defined for each goal (see Table 1 for CM goals and practices).

Getting Started

There are many ways to engage in introducing and improving CM in your organization. No way is definitely *the way* – each company must find its own way. Getting started on anything completely from scratch is not very easy; on the other hand, it is hardly ever necessary. Even if a company has the feeling that no CM is being performed at all, that is very rarely the case.

℠ CMM Integration is a service mark of Carnegie Mellon University.
® CMMI is registered in the U.S. Patent and Trademark Office.

Employees may have knowledge and practical experience of CM gained from previous jobs or during their education.

Getting the Right People

People drive changes and improvements. The success or failure of a company introducing CM may depend on the people allocated to the task. The person driving the initiative must be a fiery soul or a *true believer*. Without a burning desire to see CM at work in the company, the person in charge is prone to give up long before success is achieved. All companies have dedicated and enthusiastic people who want to do as good a job as possible.

Collecting Best Practices

It is a good idea for a company introducing CM to try and create a map of the existing knowledge on the subject within the company itself. Authorized assessments or more informal interviews with employees working on ongoing projects may be used. It is well worth the effort to analyze the information gained from these and use it to collect existing detailed information about practices and knowledge. Starting the CM initiative based on existing practices will enhance the chances of success significantly. Best practices may also be collected from external sources. A good place to start the search is in the "Configuration Management Yellow Pages" on the *CM Crossroads* Web site at <www.cmcrossroads.com/yp>.

Focus

To get CM off the ground, it is important to focus on the employees' understanding of CM and their capabilities to perform CM as on the company's need for control. This should not in any way contradict the organization's interests; at the end of the day, the company depends on the employees' ability to carry out the CM plans.

Look Ahead

At the initial introduction of CM into a company, it is important to use a stepwise approach. Begin with a few steps, and then introduce more advanced aspects in due course. A stepwise approach includes the picking of low-hanging fruit. Set up goals obtainable within a short period of time – two to three months at the most – and celebrate when the goals are reached. However, even in a stepwise approach, it is a good idea to have at least some idea of where the long-term goal is.

The First Steps

The following is a suggestion on how to start CM from virtually nothing.

CM Goals and Practices

SG 1	Establish Baselines	
	SP 1.1-1	Identify Configuration Items
	SP 1.2-1	Establish a CM System
	SP 1.3-1	Create or Release Baselines
SG 2	Track and Control Changes	
	SP 2.1-1	Track Changes
	SP 2.2-1	Control Changes
SG 3	Establish Integrity	
	SP 3.1-1	Establish CM Records
	SP 3.2-1	Perform Configuration Audits
GG 2	Institutionalize a Managed Process	
	GP 2.1	Establish an Organizational Policy
	GP 2.2	Plan the Process
	GP 2.3	Provide Resources
	GP 2.4	Assign Responsibility
	GP 2.5	Train People
	GP 2.6	Manage Configurations
	GP 2.7	Identify and Involve Relevant Stakeholders
	GP 2.8	Monitor and Control the Process
	GP 2.9	Objectively Evaluate Adherence
	GP 2.10	Review Status with Higher-Level Management

Note: SG is specific goal for the CM. GG is general goal for all key process areas. SP is specific practice. GP is general practice.

Table 1: *CM Goals and Practices*

Establish Baselines

The primary goal is to get control over the source code and corresponding objects. This may be done by first defining and assigning CM responsibilities in the organization. Appoint at least one CM person responsible as a frontrunner.

Next, define what types of objects to place under CM. These may be individual objects such as specifications, source code modules, other files, etc., or composite objects like subsystems, partial deliveries to the customer, or entire systems. For all these types of objects, a convention for unique identification must be defined, and who has the authority over the objects (e.g., who can approve them) must be defined.

Following this, identify what has already been produced, if anything. The found objects must be stored in a controlled library with relevant information; they become configuration items. A configuration item is any intermediate working product, product component, or product that is placed under CM. Composite items of what have already been produced must also be defined, along with specifying how they are composed.

Lastly, define the rules for making releases of configuration items and to register who has which releases.

Track and Control Changes

When developing and maintaining a product, changes are inevitable. The purpose of change control is to be fully in control of all change requests and all implemented changes; a formal change control must be established for the items now under CM.

Life cycles for incidents and changes must be defined. An incident is every (significant) unplanned event observed (during test), and requiring further investigation according to British Standard 7925-1. The initiation of change control is the occurrence of an incident. It may, for instance, be a wrong formulation in a document, a coding mistake found during a walkthrough, an enhancement request arising from a new idea from the customer, or a change required in the code because of the upgrade to a new version of the middleware supporting the system. All incidents must be registered, and for this a template must be produced.

A Change Control Board (CCB) is responsible for the change control of each individual configuration item, so at least one CCB must be established. The CCB is a group of people responsible for assessing, and subsequently approving or rejecting proposed changes to configuration items. *A CCB must be formed according to the*

wanted level of formality. It may consist of the author or producer of an item, a peer group, or a number of managers. A project may have several CCBs with different areas of responsibility.

When analyzing an incident, the CCB must consider the cost of implementing any changes compared to the cost of not implementing these changes. The CCB decides whether or not the incident should cause one or more changes to configuration item(s). Each change to be made should be documented in a change request issued by the CCB. Make sure all incident registrations are handled by the CCB. Most importantly *do not accept that changes are implemented based on any other input than a change request from the CCB!*

A change process is a miniature development project in itself. Each phase should be described in details stating the responsibility and specific actions. The phases are outlined in Table 2.

Quite often no independent change requests are created. However, this is not a very good idea, especially in the cases where an incident causes changes in several configuration items.

It must be ensured that the CCB approves all implemented changes. All new changed configuration items must be identified and stored in the controlled library with the corresponding information. The last task in the incident life cycle is to provide feedback on decisions to every stakeholder.

Establish Integrity

It is necessary to register data for configuration items such as names, versions, and status, and to store all incident registrations and change requests. Information kept in the CM system is a gold mine of metrics for the project. Make sure to keep relevant information available. Reports such as release notes, item status lists, item history lists, item composition lists, and trace reports should be defined. Audits are not discussed here as the responsibility for these often lies in quality assurance in modern companies.

Good Enough Is Not Always Enough

A CM system as described can be a very good beginning. It will, however, in most cases not be sufficient for a project or a company to get full profit from performing CM. A project well begun is half done, and even a minimal system provides a good starting point for the expansion of CM to a more comprehensive system in order for the benefits to grow. It is, however, important to keep in mind that the scope and the degree of formality must never exceed what is profitable for the company.

Institutionalize a Managed Process

For a key process area to reach a specific level in CMMI terms, the generic practices need to be fulfilled as well as the specific practices. The generic practices are discussed briefly below in view of CM.

Establish an Organizational Policy

An organizational policy is the responsibility of top management and defines the organization's philosophy towards CM. It must be short and high-level and include a definition of CM to be used as the basis for the CM work in the organization, a description of the CM process to be used, ways of evaluating the CM performance in the organization, and the approach to CM process improvement.

Plan the Process

Even if the CMMI did not promote planning, the starting point for good CM is in planning. If you do not plan, you plan to fail! The work with the plan deepens the understanding of the task and provides the basis for the actual performance of the work to be undertaken.

All the CM activities to be performed must be described in the CM plan for the project. Note that the CM plan may be included as a chapter in the overall project plan if that is preferred. The CM plan does not need to be an independent document, but always make sure that it is easy to recognize, and understand the connections between the CM activities and the other activities in the project. And independent of the size, do not forget to allow for the time to plan the CM.

When planning CM, the purpose, success criteria, and level of ambition must be defined. The plan must be a living document and be used – beware of write-only information. Also, record what has consciously been left out at the point of time when writing the plan. Make sure that all stakeholders, including employees on the project team, are familiar with the plan and willing to adhere to it.

A template is a great help. In many companies, templates for plans in general and even specifically for CM plans exist. Use them! A CM plan may be structured in the following way:

1. Introduction
   - 1.1 Purpose of the Plan
   - 1.2 Scope of the CM Task
   - 1.3 Vocabulary
   - 1.4 References
2. Management and Relations to the Environment
   - 2.1 Organization
     Specify how the overall organization of CM is formed, and how this fits into the rest of the project organization. Describe which CM roles are to be filled.
   - 2.2 Responsibilities
     Define the following clearly and unambiguously:
     - Who is responsible for performing which CM activity?
     - Who is responsible for approval of objects prior to placement under CM?
   - 2.3 Interface Control
     Define how the interfaces to external objects (software, hardware, etc.) are handled with regard to CM.
   - 2.4 Subcontractor Management
     Define how new versions are to be tested for approval, who is doing this, and how the deliveries from the subcontractor(s) are introduced into the project in a controlled way.

Table 2: *Change Control Phases*

Phase	Description
Creation of the incident registration.	The incident is described in the registration.
Analysis of the incident registration.	It is determined which configuration item(s) will be affected by possible changes, and the change effort is estimated.
Rejection or acceptance of the incident.	If the incident is accepted, a change request is created for each of the affected configuration items.
The change request initiates a new configuration item.	A new configuration item is identified and created, and the change is implemented. In the course of acceptance and placement in storage of the new configuration item, feedback is given to the CCB.
Closing of the change request.	The change request can be closed when the change has been implemented and accepted.
Closing of the incident registration.	The incident registration can be closed when all the the corresponding change requests are closed.

2.5 Relevant Standards
Describe which guidelines and policies the concrete project adheres to.

3. Activities
   3.1 Identification
   Describe and plan the activities concerning naming conventions for configuration items and the rest of the CM data (meta data). At the time of the planning, some of this information is typically yet unknown, e.g., module names. In this case, it must be described how and when this missing information will be provided. This section may describe how the information is stored. Detailed techniques may be placed in an appendix.
   3.2 Storage and Release
   Describe the handling of the controlled library, i.e., how and when the library is built and deployed, and how information is collected. Consider how long the various types of CM information should be kept. Finally, it may be described how necessary backup of the libraries and other information is performed. This is, strictly speaking, not a CM activity, but it is certainly in the interest of CM to ensure that backups are taken, kept in a safe place, and can restore correctly.
   3.3 Change Control
   This is possibly the section most important to get in place so that an appropriate balance between formalism and flexibility is reached. Define who has the authority to request changes to a configuration item, i.e., who are members of the CCBs for the various object types. A CCB may delegate the authority to individuals or other roles; this should be planned and documented. Define how incident registrations and change requests for various types of objects, e.g., documentation, code, support software and tools, are to be handled, that is, defining the life cycles for incidents and changes. When these procedures are established, it must be ensured that events and changes can be handled fast enough so that CM is not perceived as an unnecessary bottleneck in stressed situations such as important deliveries to customers.
   3.4 Status Reporting
   Great benefits are to be gained from performing CM; valuable information is easier to find and use. Considerations regarding status reporting provide significant input about which data should be registered and for how long the information should be kept. Use this section to define how status information about the configuration items is collected, treated, and reported on. Also define periodic reports and how ad-hoc or dynamic queries should be handled.
4. Schedule
   4.1 Tasks
   List the detailed tasks here. Make a special effort not to forget anything – avoid invisible work, i.e., work that the staff has to perform but which is not covered in the plan and the schedule. Also determine and plan required training. Document as far as possible the actual people who will be performing the related tasks. The interfaces or connections to the overall project plan should be stated.
   4.2 Milestones
   CM has a number of milestones, e.g., the CM system is ready for use for the next activity in the project, the CCB(s) is (are) established, deliveries of subsystem and systems, and product release. List the milestones and state in which phases there is a need for which roles and resources, preferably with a reference to the overall project plan.
   4.3 Diagrams and Charts
   Support the planning descriptions with diagrams and charts. Follow up on the schedules, i.e., clearly mark on diagrams and charts where the project is, and frequently re-plan so the plan reflects reality.
5. Tools, Techniques, and Methods
   Depending on the capability level of the organization, this chapter may be the easiest or the hardest part of the plan to write. If there is no central place to get help, like an organization-wide quality system, and the descriptions in this chapter are to be useful at all, the chapter will be hard to write, possibly fairly big, and it may take a long time. If, on the other hand, general processes exist, simple references and a few descriptions of tailoring may be sufficient. In any case, do not underestimate the importance of this information, and the work involved in providing it.

Provide Resources, Assign Responsibility, Train People

The provision of resources, assignment of responsibility, and training of people are parts of the planning activity. The plan must reflect the actual recourses available for the tasks.

Manage Configurations

This means that products from the CM process must be placed under CM. It is appropriate to place the CM plan under CM, so it should adhere to the general rules of unique identification of objects and be stored in a controlled library, as well as having all changes be controlled in the specified way.

Identify and Involve Relevant Stakeholders

Almost everybody involved in the specification, development, usage, and maintenance of a system is a stakeholder in CM.

Table 3: *Processes Needed for CM as Defined By CMMI*

Processes Needed for CM as Defined By CMMI
For Establishing Baselines: • Convention(s) for unique identification. • Convention(s) for identification of single items and composite items. • Procedure(s) for registration of information about each configuration item. • Procedure(s) for placement in storage and related update of information. • Procedure(s) for release of items. • Template(s) for item approval registration. • Template(s) for release request.
For Tracking and Controlling Changes: • Convention(s) for formation of different types of CCBs. • Definition of responsibility for each type of CCB. • Description of the change control process. • Procedures forming the life cycles for incidents and changes. • Template(s) for incident registration. • Template(s) for change request.
For Establishing Integrity: • Procedure(s) for production of available reports. • Procedure(s) for ad-hoc extracts of information. • Procedure for audit. • Templates for the reports that the CM system is expected to produce.

Make a list and make it comprehensive – remember everybody.

Monitor and Control the Process

To be able to monitor and control processes, they must be understood and defined. The processes needed for CM as defined by CMMI in terms of procedures, conventions, and templates are listed in Table 3.

The fact that CM is performed during the entire lifetime of a product, for a number of different types of objects, and under various circumstances, poses specific requirements on the process descriptions for CM. It may be necessary to produce more variants of some of the processes, depending on the types of configuration items to handle, and/or the degree of formalism required. Types of configuration items, to mention a few, may be requirement specifications, source code, and plans.

Metrics for Controlling CM Performance

Tom Gilb said in "Principles of Software Engineering Management," that everything can be made measurable in a way that is better than not measuring at all [7]. Measurements may show new aspects of things you thought you knew everything about.

The following are suggestions for metrics that may be used for analyzing how CM is performed in a company. The CM processes are in focus here, not other processes and not the product. The list is by no means exhaustive.

- Number of registrations of configuration items in the CM system.
- Time interval in which the registrations have taken place.
- Time used for each individual registration.
- Incidents in connection with registrations.
- Incident rate for registrations, e.g., number of erroneous registrations per 100.
- Average time for registration.

Some of the metrics may be defined by configuration item type. Identical metrics may be defined for the following:

- Placement in storage.
- Releases.
- Handling of incident registrations.
- Handling of change requests.
- Completions of milestones defined in the CM plan.

Metrics may also be defined to include costs such as the cost of the activities.

There is no reason for collecting measurements if no one is going to analyze them or act according to the analysis results. Measurements must be collected over time for the balance point and the variation to be calculated, for example.

Process control is to find reasons for variations from the normal. Control over processes is gained by constantly analyzing new measurements in relation to the expected values, so processes that start to behave in an unexpected way can be identified. The reason for the unexpected behavior must be investigated and possible irregularities identified.

An unexpected behavior may be seen by a measurement that suddenly lies far outside the normal variation. This could be to either the positive or negative side. An example might be the handling time for an incident that is suddenly far faster than normal – is this because the CCB did not do their work properly, is it an incident that they have handled a lot of times before, or is there a totally different reason?

Process improvement is to change the reasons why something is considered normal. In conducting process improvement, you may ask yourself questions like the following:

- Why is the balance point where it is?
- How may the balance point be moved?
- Why does the variation have the size it has?
- How may the variation be decreased?

A balance point may be the average time it takes for an incident registration of a certain type to be handled. It may be worthwhile finding out if there are bottlenecks in the handling process that might be eliminated to decrease the average handling time.

Objectively Evaluate Adherence and Review Status With Upper Management

These practices are the responsibility of quality assurance and management and will not be discussed here.

Good Luck With CM

CM is not easy! If you think it is, you will be unable to solve the CM task in a professional way. CM is not difficult! All you have to do is to do it; if you understand the discipline, it is much easier to specify and plan the task so that it fulfills its purpose and becomes manageable.◆

References

1. Del Duca, Gianfranco. "Introduction of CM: Gaining a Competitive Edge." <www.esi.es/VASIE/Reports/All/24151/14.pdf>.
2. Soro, Roberto. "Applying GQM to Assess CM Practice for Better Interbank Services." <www.esi.es/VASIE/ Reports/All/24151/sia.pdf>.
3. Garella, Alberto, and Giuseppina Tallo. "Configuration and Change Management to Rising Quality of Service." <www.esi.es/VASIE/Reports/All/24151/istiserv.pdf>.
4. Vidvei, Tor. "Introduction of a CM in Very Small Organizations." <www.esi.es/VASIE/Reports/All/24151/40.pdf>.
5. Roald, Helge M. "Introduction of a Common CM Framework." <www.esi.es/VASIE/Reports/All/24151/52.pdf>.
6. CMMI Product Team. Capability Maturity Model® Integration (CMMI SM), V.1.1. Pittsburgh, PA: Software Engineering Institute, Mar. 2002 <www.sei.cmu.edu/cmmi/models/model-components-word.html>.
7. Gilb, Tom. Principles of Software Engineering Management. Addison-Wesley, 1988.

Note

1. A description of the SISSI project, "Process Improvement Experiment Final Report," is at <www.esi.es/en/Projects/VASIE/Reports/All/21379/Report/21379.pdf>.

About the Author

Anne Mette Jonassen Hass, Mc.Sc.C.E., has been a consultant for DELTA since 1995, where she is now partner. She has 25 years experience in information technology (IT) and has performed more than 40 assessments. Hass has an Information Systems Examinations Board certificate in Software Testing Foundation and Practitioner, and is a certified BOOTSTRAP V.3.0 and Capability Maturity Model® Integration assessor. She is author of "CM Principles and Practice" and "Requirements Development and Management," and developed "Process Contest," a game that teaches IT concepts in a relaxed and entertaining way.

DELTA
Venlighedsvej 4
2970 Hoersholm
Denmark
Phone: +45 72 19 40 00
Direct: +45 72 19 44 23
Fax: +45 72 19 40 01
E-mail: amj@delta.dk

Software Plumbing

Infrastructure. It's easy to ignore, it's often buried, and can tend to be dirty. I learned about infrastructure firsthand while in college. I paid for my dabbling in the dark arts of electronics with a great job on the maintenance crew for a large downtown shopping mall and high-rise. This place was packed with infrastructure. My most memorable lesson came when old Ben had me perched on the top of a really tall ladder armed with an overweight pipe wrench. Ben claimed to have personally known Gen. Patton and served him well in his engineering corps. He shared some great stories on the general's tactical use of engineering, but that's another story. In this instance, Ben was standing below – and some distance away from – my perch, directing his cadet plumber.

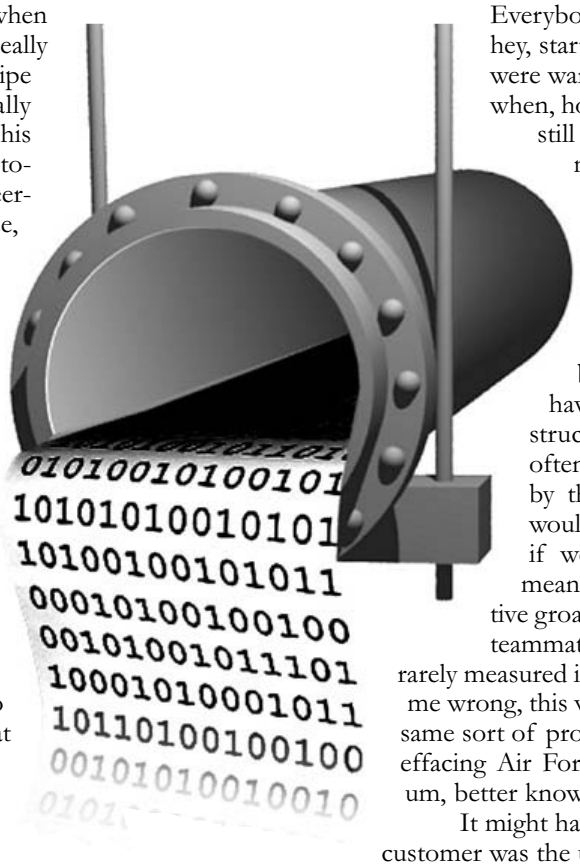


Illustration by Keith Gregersen.

It was a bit smelly, but I was doing fine on my task when my outlook changed dramatically – through the open pipe I heard a toilet flush! I was never very athletic, but now I had a vision that made me cover 15 feet before Ben could get a word out. Ben was not amused at my desertion, and sent me back up the ladder. Once I realized my fears of a gushing pipe were unfounded, Ben could see that it was time I learned about infrastructure. "Tony," he said, "you need to know the two rules of plumbing, the first rule being that water flows downhill." I quickly perceived that this first rule was the only reason I was still smiling.

"So what's the second rule?" I asked.

"Don't bite your fingernails!" he said.

Truth be known, it was Ben and his troops that kept the folks upstairs from biting their nails. The whole city block was cooled, heated, lubricated, lighted, and generally functioned because Ben and I were at work. I left Ben's tutelage to begin my career in software and before long found that even software had something akin to infrastructure. The analogy isn't perfect, but just like the undergirding of a building, my initial dabbling in code exposed the underpinnings of the software world, and included in that foundation I found configuration management (CM) and test.

Now, lest the hate mail flow, my reference of CM, test, and sewage in the same article is simply a matter of artistic contrivance to show that these disciplines – like infrastructure in the physical world – hold up the software universe. For example, I heard of one company whose CM was so neglected that the developers literally lost the baseline for their software. Imagine that! They climbed out of this hole by begging the productions guys for a copy of their own software – that's like Julia Child calling Random House for her macaroni and cheese recipe. Talk about missing the essentials!

I've never witnessed such gyrations firsthand; in fact, I became an early proselyte to the ways of CM when it saved my skin. I was merrily cranking code on one of my first assignments when our team lead announced an inquisition; it seemed our latest additions to our product had rendered it, well, useless. As the least experienced on the team, I was the guilty one. But we all marched out to our integration stand where I witnessed the marvel of CM as the build was reconstructed change by change. Everybody wanted to get this over with so, hey, start with the new guy first. My cohorts were warming up a nice selection of ridicule when, horrors – for them anyway – the thing still worked with my code in it. This meant, of course, that someone else was to blame and the cold hand of CM was about to expose him. Which it did, and I don't recall my team lead ever convening another such inquisition.

All right, so exposing the boss was a bit messy, but it would have been a real battle without the structure of CM. Being a test engineer often put me in similarly tight spots, but by then I had learned to enjoy it. We would work endless hours in the lab and if we ever showed up in the office, it meant trouble. You could feel the collective groan as we strode in and toyed with our teammates. Yes, indeed, personal pride is rarely measured in the cost of zero defects. Don't get me wrong, this was gentle professional ribbing – the same sort of professionalism that produced the self-effacing Air Force Acronym Reference Compendium, better known in testing circles as the AFARC.

It might have been a bit rough at times, but the customer was the unknowing beneficiary. Working the infrastructure of software may feel sometimes like working the nameless unseen systems in Ben's building, but that never seemed to bother Ben. He knew.

— Tony Henderson
Software Technology Support Center
tony.henderson@hill.af.mil

Can You BACKTALK?

Here is your chance to make your point, even if it is a bit tongue-in-cheek, without your boss censoring your writing. In addition to accepting articles that relate to software engineering for publication in CROSSTALK, we also accept articles for the BACKTALK column. BACKTALK articles should provide a concise, clever, humorous, and insightful perspective on the software engineering profession or industry or a portion of it. Your BACKTALK article should be entertaining and clever or original in concept, design, or delivery. The length should not exceed 750 words.

For a complete author's packet detailing how to submit your BACKTALK article, visit our Web site at <www.stsc.hill.af.mil>.

CrossTalk / 309 SMXG/MXDB
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737